Michele Mariotti
Mirko Bruschi

Sistema di monitoraggio del traffico tramite Zeek e Tensorflow

In questo progetto è stata utilizzata una macchina con sistema operativo Windows 11, WSL v2 e una macchina virtuale con Ubuntu Server 24.04.1 LTS.
L'obiettivo del sistema di monitoraggio è individuare se un pacchetto è di tipo TLS, in base ad un certo numero di bit del payload. WINDOWS 11 e UBUNTU SERVER sono collegati alla rete tramite bridge, WSL e WINDOWS 11 sono collegati tra loro tramite NAT eseguito dal pc windows.
Si riporta lo schema di collegamento delle macchine che sarà utile per la trattazione successiva.

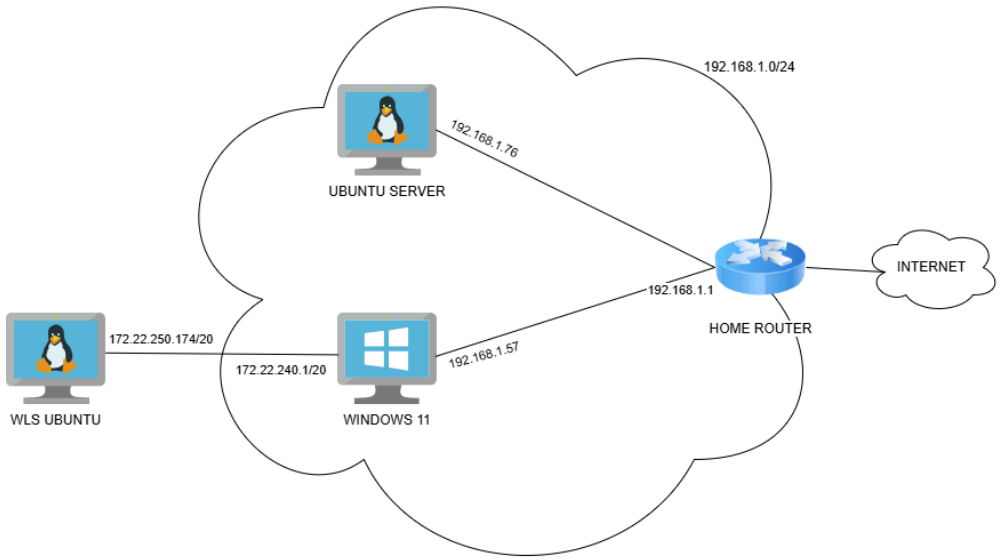


Immagine di esempio: gli indirizzi ip delle interfacce collegate al router possono cambiare in base al metodo di assegnazione degli indirizzi ed al network ID.

PARTE 1: Configurare Zeek

Zeek è un analizzatore di traffico di rete open source che è stato originariamente sviluppato presso il National Laboratory Lawrence Berkeley. È uno strumento potente e flessibile che può essere utilizzato per una varietà di attività di sicurezza della rete, tra cui il rilevamento delle intrusioni e l'analisi del traffico di rete.
Zeek è scritto nel linguaggio di scripting Zeek ed è noto per la sua capacità di gestire grandi quantità di traffico di rete. È altamente personalizzabile, consentendo agli utenti di scrivere i propri script per analizzare il traffico di rete in modi specifici.

In questo esempio, per semplicità, è stata realizzata una sonda software che raccoglie il traffico della macchina che la ospita (Ubuntu Server 24.04.1 LTS). Per generalizzare sarebbe sufficiente collegare la macchina alla porta di mirroring di un router o di uno switch e monitorare l'interfaccia su cui è realizzato il collegamento.

In seguito viene riportata la procedura da svolgere sulla macchina UBUNTU SERVER:

1. Installare i pre-requisiti:
   ```
   apt-get update
   apt-get install -y --no-install-recommends g++ cmake make libpcap-dev
   ```

2. Installare Zeek 6.0 come indicato sulla guida ufficiale (https://docs.zeek.org/en/master/install.html):
   ```
   2.1 echo 'deb
   http://download.opensuse.org/repositories/security:/zeek/xUbuntu_22.04/ /' | sudo
   tee /etc/apt/sources.list.d/security:zeek.list
   2.2 curl -fsSL
   https://download.opensuse.org/repositories/security:zeek/xUbuntu_22.04/Release.key
   | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/security_zeek.gpg > /dev/null
   2.3 sudo apt update
   2.4 sudo apt install zeek-6.0
   ```
3. Aggiungere Zeek al PATH:
   3.1 Scoprire posizione di Zeek: which zeek
   3.2 Inserire alla fine del file ~/.bashrc le righe:
   ```
   export PATH="$PATH:/opt/zeek/bin" (utente standard)
   export PATH="/opt/zeek/bin:$PATH" (utente superuser)
   ```
   3.3 Applicare le modifiche: source ~/.bashrc
   3.4 Verificare l'installazione: zeek --version
4. Creare in /opt/zeek/share/zeek/site un nuovo file (comms.zeek):

```
module Tensorflow;

# definisce un nome per l'evento generico e quali sono i dati raccolti
global content: event(orig_h:addr, orig_p:port, resp_h:addr,
    resp_p:port, content:string);

# inizializza il broker
event zeek_init()
{
      Broker::subscribe("tensorflow/content");
      Broker::listen("192.168.1.76", 9999/tcp);
      Broker::auto_publish("tensorflow/content", content);
}

# funzione che pubblica l'evento "content", contenente informazioni
sulla connessione
function output(c:connection, bytes:string)
{
      Broker::publish("tensorflow/content", content, c$id$orig_h,
```

```
        c$id$orig_p, c$id$resp_h, c$id$resp_p, bytes);
}


# analizza i pacchetti TCP e, se il pacchetto è il primo della
connessione (SYN, seq=1, con payload), lo invia
event tcp_packet(c:connection, is_orig:bool, flags:string, seq:count,
    ack:count, len:count, payload:string)
{
      if(is_orig && "S" in c$history && seq==1 && |payload|>0)
      {
            output(c, string_to_ascii_hex(payload));
      }
}
```

5. Aggiungere al file local.zeek in /opt/zeek/share/zeek/site la riga `@load comms` per avviare automaticamente lo script all'avvio di zeek
6. Modificare il file node.cfg in /opt/zeek/etc sostituendo il valore di interface con il nome dell'interfaccia su cui eseguire il monitoraggio

PARTE 2: Installare Broker e configurare la rete

Broker è una libreria per la comunicazione publish/subscribe, che consente lo scambio di dati tra istanze di Zeek o altri sistemi. Supporta diversi tipi di dati basati sul modello di Zeek e fornisce strumenti per la gestione degli eventi distribuiti.
In questo caso è stata installata all'interno di WSL (Ubuntu).
Vengono riportati i passaggi da svolgere:

1. Installare WSL Ubuntu da Windows Powershell: `wsl --install Ubuntu`
2. Avviare WSL ed eseguire le successive azioni su questa macchina
3. Installare i prerequisiti: `sudo apt install python3-pip python3-virtualenv cmake libssl-dev`
4. Clonare la repository GitHub di broker:
   `git clone --recursive https://github.com/zeek/broker.git`
5. creare un nuovo ambiente virtuale (venv): `virtualenv -p python3 /home/$USER/venv`
   per attivarlo utilizzare il comando: `source /home/$USER/venv/bin/activate`
6. Attivare l'ambiente virtuale
7. Entrare nella cartella broker scaricata al punto 4 ed eseguire all'interno del venv
   `./configure --prefix=/home/$USER/ --python-prefix=$(python3 -c 'import sys; print(sys.exec_prefix)')`
8. Avviare l'installazione con `make install`
9. Verificare l'installazione con `python3 -c 'import broker; print(broker.__file__)'`
10. WSL si connette alla rete tramite un router virtuale con NAT gestito da Windows. Questa configurazione permette ai pacchetti provenienti da WSL di raggiungere qualsiasi dispositivo presente in LAN e di accedere a Internet. Tuttavia, per impostazione predefinita, non è possibile il contrario: i dispositivi esterni alla rete locale o su Internet non possono avviare connessioni dirette verso WSL. Nel caso specifico in cui WSL debba ricevere pacchetti da una macchina virtuale presente

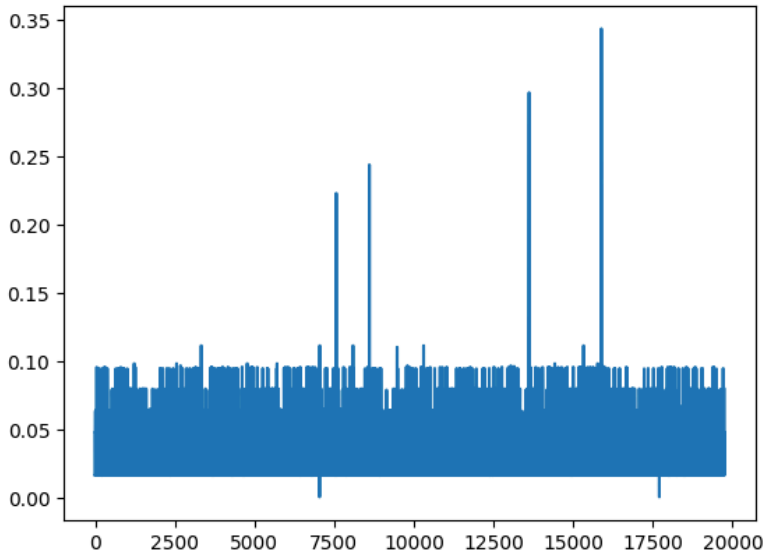
sulla stessa rete, è necessario configurare una regola di port forwarding sul router virtuale di Windows. Questa regola ha lo scopo di "inoltrare" i pacchetti in arrivo a Windows, destinati a una specifica porta, verso l'indirizzo IP e la porta corrispondente all'interno di WSL. Nel nostro caso vogliamo inoltrare a WSL tutti i pacchetti che hanno porta 9999. Per far questo da Windows Powershell eseguiamo: `netsh interface portproxy add v4tov4 listenport=9999 listenaddress=0.0.0.0 connectport=9999 connectaddress=172.22.250.174`

PARTE 3: Realizzazione dei modelli

autoencoderTLS.py

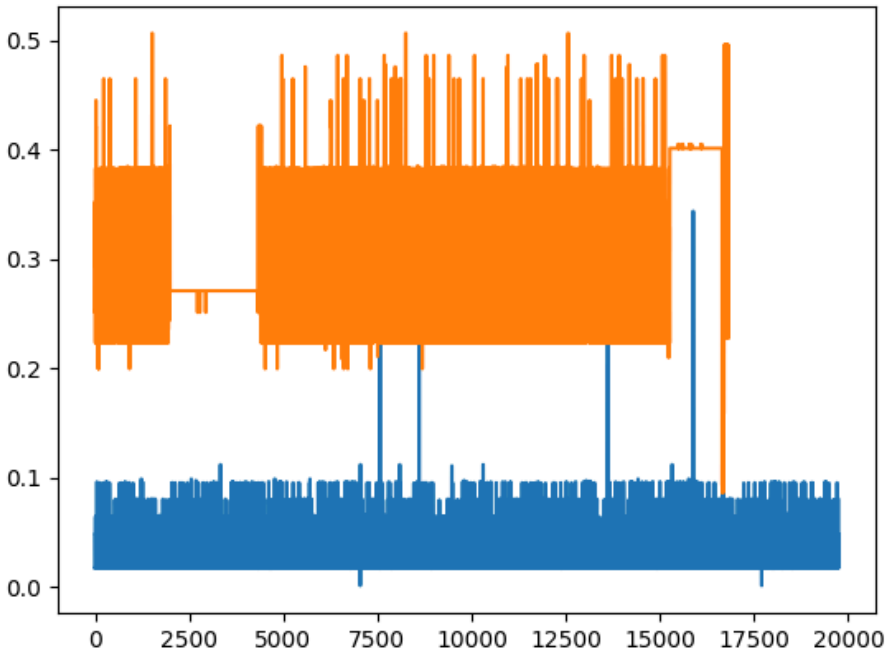
```
import tensorflow as tf
from tensorflow.keras import Sequential, layers
from tensorflow.keras.layers import Dense
import numpy as np



MAX_LENGTH = 8 #numero di bytes del payload che tengo, dopo aver provato vari valori 8 è quello che fornisce la miglior performance
def content_to_features(data, number_of_bytes = MAX_LENGTH):
    x = data.ljust(number_of_bytes * 2, "0")
    x = x[:(number_of_bytes * 2)]

    byte_values = []
    for i in range(0, len(x), 2):
        byte_values.append(chr(int(x[i:i+2], 16)))
    bits = []
    for byte in byte_values:
        bits.append([(ord(byte) & (1<<i)) >>i for i in [7, 6, 5, 4, 3, 2, 1, 0]])

    return list(np.array(bits).astype(np.uint8).flat)



with open('./training_data/TLS', 'r') as f:
    raw_data = [content_to_features(line) for line in f.readlines()]

x_data = np.array(raw_data).astype(np.uint8)
```

```
#print(x_data.shape)



x_data = tf.random.shuffle(x_data, seed=100)

VALIDATION_SET_SIZE = round(0.1 * x_data.shape[0])  # 10%
TEST_SET_SIZE = round(0.2 * x_data.shape[0])  # 20%
TRAIN_SET_SIZE = x_data.shape[0] - VALIDATION_SET_SIZE - TEST_SET_SIZE  # 70%

x_train, x_val, x_test = tf.split(x_data, [TRAIN_SET_SIZE,
VALIDATION_SET_SIZE, TEST_SET_SIZE])



input_dimensions = MAX_LENGTH * 8   #numero di input = numero di bit

model = Sequential()
model.add(Dense(input_dimensions, input_shape=(input_dimensions,)))
model.add(Dense(input_dimensions // 2, activation='relu'))
model.add(Dense(input_dimensions // 4, activation='relu'))
model.add(Dense(input_dimensions // 4, activation='relu'))
model.add(Dense(input_dimensions // 2, activation='relu'))
model.add(Dense(input_dimensions, activation='relu'))
# funzione di attivazione relu plausibile in quando ho dei valori binari



model.compile(optimizer='adam', loss='mae')
model.fit(x_train, x_train, batch_size=128, epochs=10)  #labels sono sempre le
features stesse nell'autoencoder



import matplotlib.pyplot as plt
plt.plot(tf.losses.mae(x_val, model.predict(x_val)))
plt.show()
```

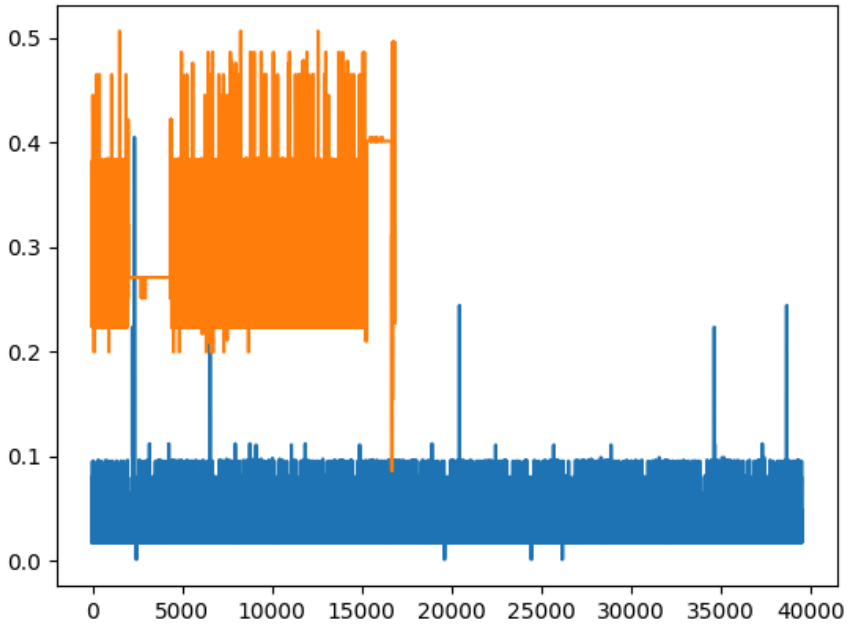



```
with open('./training_data/HTTP', 'r') as f:
    raw_data = [content_to_features(line) for line in f.readlines()]



http_data = np.array(raw_data).astype(np.uint8)
#print(http_data.shape)



""" porta 22: SSH (Secure Shell)
porta 53: DNS
porta 1119: servizi di gaming online """
ports = [22, 53, 1119]

raw_data = []

for port in ports:
    file_path = './training_data/' + str(port)
    with open(file_path, 'r') as f:
        raw_data.extend([content_to_features(line) for line in f.readlines()])

other_data = np.array(raw_data).astype(np.uint8)
#print(other_data.shape)
```

```
not_tls_data = np.concatenate((http_data, other_data), axis=0)



#fase tuning iperparametro attraverso validation set
plt.plot(tf.losses.mae(x_val, model.predict(x_val)))
plt.plot(tf.losses.mae(not_tls_data, model.predict(not_tls_data)))
plt.show()
```



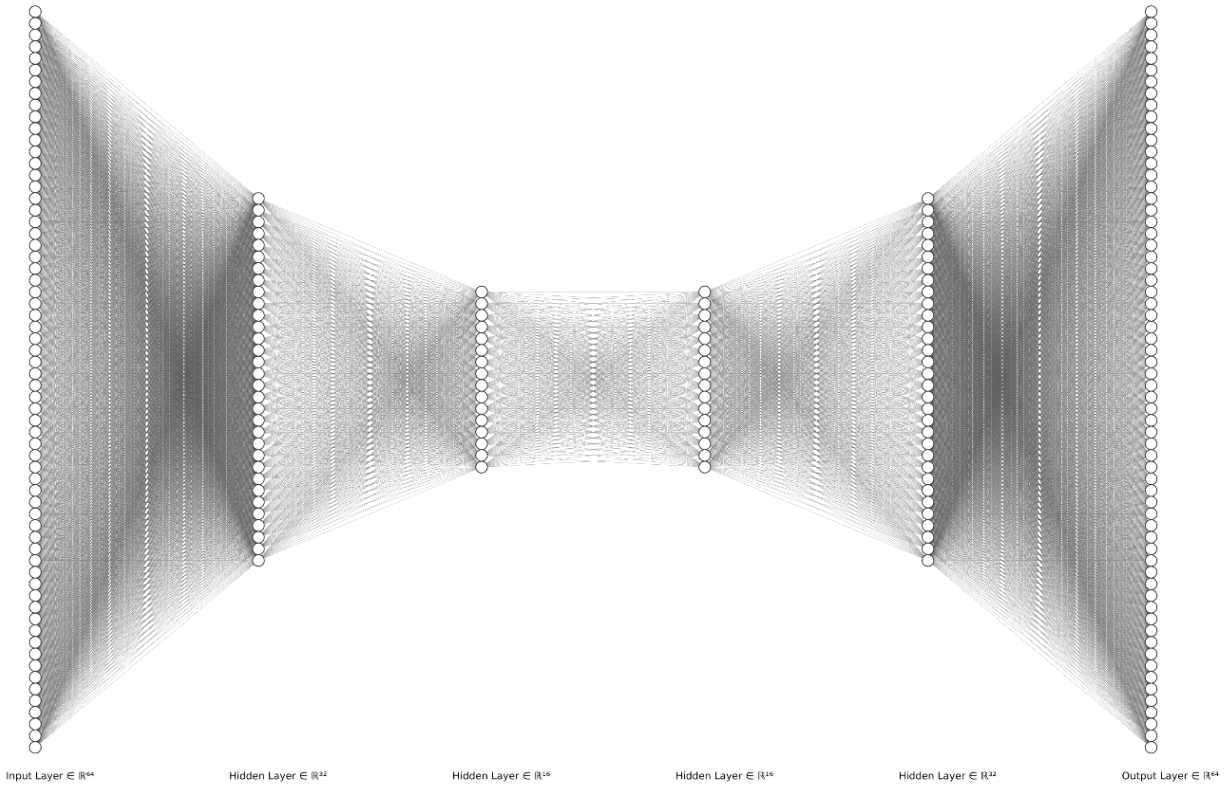

```
#soglia individuata
THRESHOLD = 0.20



#fase test
loss_x_test = tf.losses.mae(x_test, model.predict(x_test))
loss_not_tls = tf.losses.mae(not_tls_data, model.predict(not_tls_data))

plt.plot(loss_x_test)
plt.plot(loss_not_tls)
plt.show()
```




```
FN = np.sum(loss_x_test > THRESHOLD)
TN = np.sum(loss_not_tls > THRESHOLD)
TP = len(loss_x_test) - FN
FP = len(loss_not_tls) - TN

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * TP / (2 * TP + FP + FN)

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1_score)
```

```
Accuracy:  0.999538204681894
Precision:  0.99949359396364
Recall:  0.9998480243161094
F1 Score:  0.9996707777243143
```

```
#riaddestra con tutti i dati (compreso validation e test set) prima di
esportare
model.fit(x_data, x_data, batch_size=128, epochs=10)
model.save('autoencoderTLS_model.keras')
```



Schema autoencoder: in input abbiamo 64 bit che passano attraverso 4 hidden layers, di rispettivamente 32, 16, 16 e 32 bit, per poi essere ricostruiti nell'output.

Per addestrare il modello è stato utilizzato del traffico TLS, successivamente per effettuare il tuning degli iperparametri e valutare la performance è stato utilizzato anche del traffico non TLS, in particolare HTTP, SSH (Secure Shell), DNS e di servizi di gaming online.

Attraverso tuning manuale sul validation set sono stati trovati i migliori valori per gli iperparametri MAX_LENGTH e THRESHOLD. THRESHOLD verrà utilizzato successivamente per classificare i pacchetti, nella sua definizione è stato tenuto in considerazione che i dati reali avranno sicuramente una loss maggiore rispetto a quelli del validation e test set (molto simili a quelli utilizzati per addestrare il modello).

PARTE 4: Classificare i pacchetti in real-time utilizzando i modelli

La classificazione viene eseguita dal file receiver.py:

```
import sys
import importlib
from time import time
sys.path.append('/home/studente/venv/lib/python3.12/site-packages'
)
import broker
importlib.reload(broker)
import numpy as np
from tensorflow import keras
import tensorflow as tf


MAX_LENGTH = 8 #numero di bytes del payload che tengo


# load model from file
TLSmodel =
keras.models.load_model('tf_models/autoencoderTLS_model.keras')


# set threshold
THRESHOLD_TLS = 0.20
```

```
def content_to_features(data, number_of_bytes = MAX_LENGTH):
    x = data.ljust(number_of_bytes * 2, "0")
    x = x[:(number_of_bytes * 2)]
    byte_values = []
    for i in range(0, len(x), 2):
        byte_values.append(chr(int(x[i:i+2], 16)))
    bits = []
    for byte in byte_values:
        bits.append([(ord(byte) & (1<<i)) >>i for i in [7, 6, 5,
4, 3, 2, 1, 0]])
    return list(np.array(bits).astype(np.uint8).flat)




endpoint = broker.Endpoint()
subscription = endpoint.make_subscriber("tensorflow/content")
status_subscription = endpoint.make_status_subscriber(True)
endpoint.peer("192.168.1.76", 9999)


# Attendi lo stato PeerAdded
while True:
    status = status_subscription.get()  # Questo si blocca fino a
quando un messaggio è disponibile
    if isinstance(status, broker.Status):
        print(f"Received status: {status.code()} - {status}")
        if status.code() == broker.SC.PeerAdded:
            print("Connected!")
            break
    else:
        print(f"Unexpected type for status: {type(status)}")
        sys.exit(1)

while True:
```

```
    (tag, data) = subscription.get()
    (src, sport, dst, dport, content) =
broker.zeek.Event(data).args()
    # classify content

    reconstruction_TLS =
TLSmodel.predict(np.array([content_to_features(content)]))
    loss_TLS = tf.keras.losses.mae(reconstruction_TLS,
content_to_features(content))

    if(loss_TLS > THRESHOLD_TLS):
        print(f"SRC: {src}:{sport} - DST: {dst}:{dport} - TYPE:
Other")
    else:
        print(f"SRC: {src}:{sport} - DST: {dst}:{dport} - TYPE:
TLS")
```

Al codice mostrato aggiungiamo una sintetica spiegazione. Questo script utilizza l'autoencoder per classificare pacchetti di rete come TLS o altro tipo. Esegue alcuni passaggi fondamentali:

1. Importa le librerie necessarie e carica il modelli pre-addestrato.
2. La funzione content_to_features() converte i dati del payload del pacchetto in un array di MAX_LENGHT*8 bit
3. Configura una connessione tramite broker:
    a. Si connette a un peer su 192.168.1.76:9999
    b. Sottoscrive al topic "tensorflow/content"
    c. Attende la conferma della connessione
4. Fino a quando non viene terminata l'esecuzione:
    a. Riceve pacchetti di rete
    b. Estrae il contenuto e altri metadati (source, destination, ports)
    c. Usa il modello per analizzare il contenuto
    d. Calcola l'errore di ricostruzione (loss)
    e. Classifica il pacchetto confrontando la loss con la soglia prestabilita:
        - se la loss è sotto la soglia stabilita => pacchetto TLS
        - altrimenti => altro tipo di pacchetto
    f. Mostra il risultato a schermo

PARTE 5: Utilizzare l’infrastruttura

In questa parte viene unito il risultato di ogni sezione per costruire il sistema di monitoraggio completo.

1. Sulla macchina virtuale UBUNTU SERVER creare una nuova cartella ed entrarci, successivamente lanciare:
   ```
   sudo su
   zeek -i enp0s3 comms.zeek -C
   ```
   dove enp0s3 è l’interfaccia da monitorare
2. Su WSL avviare l’ambiente virtuale venv ed eseguire lo script receiver.py:
   ```
   source /home/$USER/venv/bin/activate
   python receiver.py
   ```
   (se non è già stato fatto in precedenza scaricare numpy e tensorflow)
3. Attendere che venga creato il collegamento e l’inizializzazione di tensorflow, successivamente l’infrastruttura sarà operativa.