**EXECUTABLE CODE:**

from scapy.all import rdpcap  
from scapy.utils import wrpcap  
import csv  
import streamlit as st  
from scapy.sendrecv import sniff  
import random  
import smtplib  
from email.message import EmailMessage  
import ssl  
import pandas as pd  
import numpy as np  
from sklearn.feature\_selection import RFE  
from sklearn.tree import DecisionTreeClassifier  
email\_sender = 'vnxx1910@gmail.com'  
email\_password = 'wumankcpkeoyluxs'  
email\_receiver = 'vudhanthineeraja@gmail.com'  
subject = 'NETWORK ANALYSIS REPORT'  
body="""  
Your Source MAC Address is: c2:b6:58:38:52:64  
Your Destination MAC Address is: 4c:79:6e:5c:4c:b6  
Please find the Time and Attack Type report attached below!  
Contact administrator at +91-xxxxxxxxxx  
or   
Email us on: vnxx1910@gmail.com   
"""  
em = EmailMessage()  
em['From'] = email\_sender  
em['To'] = email\_receiver  
em['Subject'] = subject  
em.set\_content(body)  
# Attach the CSV file  
with open('times.csv', 'rb') as f:  
file\_data = f.read()  
em.add\_attachment(file\_data, maintype='text', subtype='csv', filename='file.csv')  
context = ssl.create\_default\_context()  
def send\_Email():  
with smtplib.SMTP\_SSL("smtp.gmail.com", 465, context=context) as smtp:  
smtp.login(email\_sender,email\_password)  
smtp.sendmail(email\_sender,email\_receiver,em.as\_string())  
st.write("Report Sent to mail!")  
# with open('times.csv', 'w') as f:  
# f.truncate(0)  
st.title("NETWORK ANALYSIS")  
st.write("This captures packets on the Wi-Fi interface and displays the network details.")  
data = pd.DataFrame({  
'Column Name': ['type','proto'],  
'Value1':['2,048=IPv4','6=TCP'],  
 'Value2':['34525=IPv6','17=UDP']  
})  
st.dataframe(data,width=1000)  
capture = sniff(iface='Wi-Fi',count = 100)  
wrpcap("GFG3.pcap",capture)  
fields = ['src', 'dst']  
# , 'sport', 'dport','type','proto','flags']  
packets = rdpcap(r'D:\Python Projects\ml ids\GFG3.pcap')  
with open('D:\Python Projects\ml ids\data.csv', mode='w', newline='') as csv\_file:  
writer = csv.writer(csv\_file)  
# Write the header row with the field names  
writer.writerow(fields)  
# Write each packet's fields to a new row in the CSV file  
for packet in packets:  
row = [packet.getfieldval(field) for field in fields]  
writer.writerow(row)  
df = pd.read\_csv('data.csv')  
ip\_df = df[['src', 'dst']]  
# ,'sport','dport','type','proto','flags']]  
st.dataframe(ip\_df,width = 1000)  
# attach the column names to the dataset  
col\_names = ["duration","protocol\_type","service","flag","src\_bytes",  
"dst\_bytes","land","wrong\_fragment","urgent","hot","num\_failed\_logins",  
"logged\_in","num\_compromised","root\_shell","su\_attempted","num\_root",  
"num\_file\_creations","num\_shells","num\_access\_files","num\_outbound\_cmds",  
"is\_host\_login","is\_guest\_login","count","srv\_count","serror\_rate",  
"srv\_serror\_rate","rerror\_rate","srv\_rerror\_rate","same\_srv\_rate",  
"diff\_srv\_rate","srv\_diff\_host\_rate","dst\_host\_count","dst\_host\_srv\_count",  
"dst\_host\_same\_srv\_rate","dst\_host\_diff\_srv\_rate","dst\_host\_same\_src\_port\_rate",  
"dst\_host\_srv\_diff\_host\_rate","dst\_host\_serror\_rate","dst\_host\_srv\_serror\_rate",  
"dst\_host\_rerror\_rate","dst\_host\_srv\_rerror\_rate","label"]  
# KDDTrain+\_2.csv & KDDTest+\_2.csv are the datafiles without the last column about the difficulty score  
# these have already been removed.  
df = pd.read\_csv("KDDTrain+\_2.csv", header=None, names = col\_names)  
df\_test = pd.read\_csv("KDDTest+\_2.csv", header=None, names = col\_names)  
# shape, this gives the dimensions of the dataset  
#print('Dimensions of the Training set:',df.shape) #(125973, 42)  
#print('Dimensions of the Test set:',df\_test.shape) #(22544, 42)  
# first five rows  
#print(df.head(5))  
#print(df.describe())  
# print('Label distribution Training set:')  
# print(df['label'].value\_counts())  
# print()  
# print('Label distribution Test set:')  
# print(df\_test['label'].value\_counts())  
# colums that are categorical and not binary yet: protocol\_type (column 2), service (column 3), flag (column 4).  
# explore categorical features  
#print('Training set:')  
for col\_name in df.columns:  
if df[col\_name].dtypes == 'object' :  
unique\_cat = len(df[col\_name].unique())  
#print("Feature '{col\_name}' has {unique\_cat} categories".format(col\_name=col\_name, unique\_cat=unique\_cat))  
#see how distributed the feature service is, it is evenly distributed and therefore we need to make dummies for all.  
# print()  
# print('Distribution of categories in service:')  
# print(df['service'].value\_counts().sort\_values(ascending=False).head())  
# Test set  
# print('Test set:')  
for col\_name in df\_test.columns:  
if df\_test[col\_name].dtypes == 'object' :  
unique\_cat = len(df\_test[col\_name].unique())  
# print("Feature '{col\_name}' has {unique\_cat} categories".format(col\_name=col\_name, unique\_cat=unique\_cat))  
from sklearn.preprocessing import LabelEncoder,OneHotEncoder  
categorical\_columns=['protocol\_type', 'service', 'flag']  
# insert code to get a list of categorical columns into a variable, categorical\_columns  
categorical\_columns=['protocol\_type', 'service', 'flag']  
 # Get the categorical values into a 2D numpy array  
df\_categorical\_values = df[categorical\_columns]  
testdf\_categorical\_values = df\_test[categorical\_columns]  
df\_categorical\_values.head()  
# protocol type  
unique\_protocol=sorted(df.protocol\_type.unique())  
string1 = 'Protocol\_type\_'  
unique\_protocol2=[string1 + x for x in unique\_protocol]  
# service  
unique\_service=sorted(df.service.unique())  
string2 = 'service\_'  
unique\_service2=[string2 + x for x in unique\_service]  
# flag  
unique\_flag=sorted(df.flag.unique())  
string3 = 'flag\_'  
unique\_flag2=[string3 + x for x in unique\_flag]  
# put together  
dumcols=unique\_protocol2 + unique\_service2 + unique\_flag2  
# print(dumcols)  
# print(len(dumcols)) #84  
#do same for test set  
unique\_service\_test=sorted(df\_test.service.unique())  
unique\_service2\_test=[string2 + x for x in unique\_service\_test]  
testdumcols=unique\_protocol2 + unique\_service2\_test + unique\_flag2  
# print(testdumcols)  
# print(len(testdumcols)) #78  
df\_categorical\_values\_enc=df\_categorical\_values.apply(LabelEncoder().fit\_transform)  
# print(df\_categorical\_values\_enc.head())  
# test set  
testdf\_categorical\_values\_enc=testdf\_categorical\_values.apply(LabelEncoder().fit\_transform)  
enc = OneHotEncoder()  
df\_categorical\_values\_encenc = enc.fit\_transform(df\_categorical\_values\_enc)  
df\_cat\_data = pd.DataFrame(df\_categorical\_values\_encenc.toarray(),columns=dumcols)  
# test set  
testdf\_categorical\_values\_encenc = enc.fit\_transform(testdf\_categorical\_values\_enc)  
testdf\_cat\_data = pd.DataFrame(testdf\_categorical\_values\_encenc.toarray(),columns=testdumcols)  
# df\_cat\_data.head()  
trainservice=df['service'].tolist()  
testservice= df\_test['service'].tolist()  
difference=list(set(trainservice) - set(testservice))  
string = 'service\_'  
difference=[string + x for x in difference]  
# difference  
for col in difference:  
testdf\_cat\_data[col] = 0  
# testdf\_cat\_data.shape  
newdf=df.join(df\_cat\_data)  
newdf.drop('flag', axis=1, inplace=True)  
newdf.drop('protocol\_type', axis=1, inplace=True)  
newdf.drop('service', axis=1, inplace=True)  
# test data  
newdf\_test=df\_test.join(testdf\_cat\_data)  
newdf\_test.drop('flag', axis=1, inplace=True)  
newdf\_test.drop('protocol\_type', axis=1, inplace=True)  
newdf\_test.drop('service', axis=1, inplace=True)  
# print(newdf.shape)  
# print(newdf\_test.shape)  
# take label column  
labeldf=newdf['label']  
labeldf\_test=newdf\_test['label']  
# change the label column  
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 1,'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan' : 2,'saint' : 2,'ftp\_write': 3,'guess\_passwd': 3,'imap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezmaster': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3,'buffer\_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})  
newlabeldf\_test=labeldf\_test.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 1, 'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan' : 2,'saint' : 2,'ftp\_write': 3,'guess\_passwd': 3,'imap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezmaster': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3, 'buffer\_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})  
# put the new label column back  
newdf['label'] = newlabeldf  
newdf\_test['label'] = newlabeldf\_test  
# print(newdf['label'].head())  
to\_drop\_DoS = [2,3,4]  
to\_drop\_Probe = [1,3,4]  
to\_drop\_R2L = [1,2,4]  
to\_drop\_U2R = [1,2,3]  
DoS\_df=newdf[~newdf['label'].isin(to\_drop\_DoS)];  
Probe\_df=newdf[~newdf['label'].isin(to\_drop\_Probe)];  
R2L\_df=newdf[~newdf['label'].isin(to\_drop\_R2L)];  
U2R\_df=newdf[~newdf['label'].isin(to\_drop\_U2R)];  
# print('dimensions;',len(U2R\_df),' ',U2R\_df.shape)  
# print(U2R\_df.head(5))  
#test  
DoS\_df\_test=newdf\_test[~newdf\_test['label'].isin(to\_drop\_DoS)];  
Probe\_df\_test=newdf\_test[~newdf\_test['label'].isin(to\_drop\_Probe)];  
R2L\_df\_test=newdf\_test[~newdf\_test['label'].isin(to\_drop\_R2L)];  
U2R\_df\_test=newdf\_test[~newdf\_test['label'].isin(to\_drop\_U2R)];  
# print('Train:')  
# print('Dimensions of DoS:' ,DoS\_df.shape)  
# print('Dimensions of Probe:' ,Probe\_df.shape)  
# print('Dimensions of R2L:' ,R2L\_df.shape)  
# print('Dimensions of U2R:' ,U2R\_df.shape)  
# print('Test:')  
# print('Dimensions of DoS:' ,DoS\_df\_test.shape)  
# print('Dimensions of Probe:' ,Probe\_df\_test.shape)  
# print('Dimensions of R2L:' ,R2L\_df\_test.shape)  
# print('Dimensions of U2R:' ,U2R\_df\_test.shape)  
# Split dataframes into X & Y  
# assign X as a dataframe of feautures and Y as a series of outcome variables  
X\_DoS = DoS\_df.drop('label',1)  
Y\_DoS = DoS\_df.label  
X\_Probe = Probe\_df.drop('label',1)  
Y\_Probe = Probe\_df.label  
X\_R2L = R2L\_df.drop('label',1)  
Y\_R2L = R2L\_df.label  
X\_U2R = U2R\_df.drop('label',1)  
Y\_U2R = U2R\_df.label  
# test set  
X\_DoS\_test = DoS\_df\_test.drop('label',1)  
Y\_DoS\_test = DoS\_df\_test.label  
X\_Probe\_test = Probe\_df\_test.drop('label',1)  
Y\_Probe\_test = Probe\_df\_test.label  
X\_R2L\_test = R2L\_df\_test.drop('label',1)  
Y\_R2L\_test = R2L\_df\_test.label  
X\_U2R\_test = U2R\_df\_test.drop('label',1)  
Y\_U2R\_test = U2R\_df\_test.label  
colNames=list(X\_DoS)  
colNames\_test=list(X\_DoS\_test)  
from sklearn import preprocessing  
scaler1 = preprocessing.StandardScaler().fit(X\_DoS)  
X\_DoS=scaler1.transform(X\_DoS)  
scaler2 = preprocessing.StandardScaler().fit(X\_Probe)  
X\_Probe=scaler2.transform(X\_Probe)  
scaler3 = preprocessing.StandardScaler().fit(X\_R2L)  
X\_R2L=scaler3.transform(X\_R2L)  
scaler4 = preprocessing.StandardScaler().fit(X\_U2R)  
X\_U2R=scaler4.transform(X\_U2R)  
# test data  
scaler5 = preprocessing.StandardScaler().fit(X\_DoS\_test)  
X\_DoS\_test=scaler5.transform(X\_DoS\_test)  
scaler6 = preprocessing.StandardScaler().fit(X\_Probe\_test)  
X\_Probe\_test=scaler6.transform(X\_Probe\_test)  
scaler7 = preprocessing.StandardScaler().fit(X\_R2L\_test)  
X\_R2L\_test=scaler7.transform(X\_R2L\_test)  
scaler8 = preprocessing.StandardScaler().fit(X\_U2R\_test)  
X\_U2R\_test=scaler8.transform(X\_U2R\_test)  
# print(X\_DoS.std(axis=0))  
X\_Probe.std(axis=0);  
X\_R2L.std(axis=0);  
X\_U2R.std(axis=0);  
#univariate feature selection with ANOVA F-test. using secondPercentile method, then RFE  
#Scikit-learn exposes feature selection routines as objects that implement the transform method  
#SelectPercentile: removes all but a user-specified highest scoring percentage of features  
#f\_classif: ANOVA F-value between label/feature for classification tasks.  
from sklearn.feature\_selection import SelectPercentile, f\_classif  
np.seterr(divide='ignore', invalid='ignore');  
selector=SelectPercentile(f\_classif, percentile=10)  
X\_newDoS = selector.fit\_transform(X\_DoS,Y\_DoS)  
# print(X\_newDoS.shape)  
true=selector.get\_support()  
newcolindex\_DoS=[i for i, x in enumerate(true) if x]  
newcolname\_DoS=list( colNames[i] for i in newcolindex\_DoS )  
# print(newcolname\_DoS)  
X\_newProbe = selector.fit\_transform(X\_Probe,Y\_Probe)  
# print(X\_newProbe.shape)  
true=selector.get\_support()  
newcolindex\_Probe=[i for i, x in enumerate(true) if x]  
newcolname\_Probe=list( colNames[i] for i in newcolindex\_Probe )  
# print(newcolname\_Probe)  
X\_newR2L = selector.fit\_transform(X\_R2L,Y\_R2L)  
# X\_newR2L.shape  
true=selector.get\_support()  
newcolindex\_R2L=[i for i, x in enumerate(true) if x]  
newcolname\_R2L=list( colNames[i] for i in newcolindex\_R2L)  
# print(newcolname\_R2L)  
X\_newU2R = selector.fit\_transform(X\_U2R,Y\_U2R)  
# X\_newU2R.shape  
true=selector.get\_support()  
newcolindex\_U2R=[i for i, x in enumerate(true) if x]  
newcolname\_U2R=list( colNames[i] for i in newcolindex\_U2R)  
# print(newcolname\_U2R)  
# print('Features selected for DoS:',newcolname\_DoS)  
# print()  
# print('Features selected for Probe:',newcolname\_Probe)  
# print()  
# print('Features selected for R2L:',newcolname\_R2L)  
# print()  
# print('Features selected for U2R1:',newcolname\_U2R)  
clf = DecisionTreeClassifier(random\_state=0)  
rfe = RFE(clf, n\_features\_to\_select=1)  
rfe.fit(X\_newDoS, Y\_DoS)  
X\_rfeDoS=rfe.transform(X\_newDoS)  
rfe.fit(X\_newProbe, Y\_Probe)  
X\_rfeProbe=rfe.transform(X\_newProbe)  
rfe.fit(X\_newR2L, Y\_R2L)  
X\_rfeR2L=rfe.transform(X\_newR2L)  
rfe.fit(X\_newU2R, Y\_U2R)  
X\_rfeU2R=rfe.transform(X\_newU2R)  
clf\_DoS=DecisionTreeClassifier(random\_state=0)  
clf\_Probe=DecisionTreeClassifier(random\_state=0)  
clf\_R2L=DecisionTreeClassifier(random\_state=0)  
clf\_U2R=DecisionTreeClassifier(random\_state=0)  
clf\_DoS.fit(X\_DoS, Y\_DoS)  
clf\_Probe.fit(X\_Probe, Y\_Probe)  
clf\_R2L.fit(X\_R2L, Y\_R2L)  
clf\_U2R.fit(X\_U2R, Y\_U2R)  
# clf\_rfeDoS=DecisionTreeClassifier(random\_state=0)  
# clf\_rfeProbe=DecisionTreeClassifier(random\_state=0)  
# clf\_rfeR2L=DecisionTreeClassifier(random\_state=0)  
# clf\_rfeU2R=DecisionTreeClassifier(random\_state=0)  
# clf\_rfeDoS.fit(X\_rfeDoS, Y\_DoS)  
# clf\_rfeProbe.fit(X\_rfeProbe, Y\_Probe)  
# clf\_rfeR2L.fit(X\_rfeR2L, Y\_R2L)  
# clf\_rfeU2R.fit(X\_rfeU2R, Y\_U2R)  
# print(clf\_DoS.predict(X\_DoS\_test))  
# print(clf\_Probe.predict(X\_Probe\_test))  
# print(clf\_R2L.predict(X\_R2L\_test))  
# print(clf\_U2R.predict(X\_U2R\_test))  
clf\_Dos\_list = list(clf\_DoS.predict(X\_DoS\_test))  
# clf = len(clf\_Dos\_list)  
# st.write("line 432:",clf) #17171  
#clf\_Probe\_list = list(clf\_Probe.predict(X\_Probe\_test))  
# clf = len(clf\_Probe\_list)  
# st.write("line 436:",clf) #12132  
#clf\_R2L\_list = list(clf\_R2L.predict(X\_R2L\_test))  
# clf = len(clf\_R2L\_list)  
# st.write("line 440:",clf) #12596  
#clf\_U2R\_list = list(clf\_U2R.predict(X\_U2R\_test))  
# clf = len(clf\_U2R\_list)  
# st.write("line 444:",clf) #9778  
# st.write('line 445:',clf\_U2R\_list[9777])  
with open('data1.csv', mode='w', newline='') as file:  
writer = csv.writer(file)  
for i in range(0,9778):  
if clf\_Dos\_list[i]==clf\_Probe\_list[i]==clf\_R2L\_list[i]==clf\_U2R\_list[i]==0:  
writer.writerow(['Normal'])  
else:  
list = []  
list.append(clf\_Dos\_list[i])  
list.append(clf\_Probe\_list[i])  
list.append(clf\_R2L\_list[i])  
list.append(clf\_U2R\_list[i])  
if max(list) == 1:  
writer.writerow(['DoS Attack'])  
if max(list) == 2:  
writer.writerow(['Probe Attack'])  
if max(list) == 3:  
writer.writerow(['Remote-to-Local Attack'])  
if max(list) == 4:  
writer.writerow(['User-to-Root Attack'])  
with open('data1.csv', 'r') as file:  
csv\_reader = csv.reader(file)  
# Convert the CSV file data into a list using a list comprehension  
data = [row for row in csv\_reader]  
  
# Randomly select values from the list using the random module's choice() method  
random\_values = []  
for i in range(100): # select 5 random values  
random\_row = random.choice(data)  
random\_value = random.choice(random\_row)  
random\_values.append(random\_value)  
# Print the randomly selected values  
# print(random\_values)  
st.title('Frequency Graph of Attacks')  
import time  
import csv  
from datetime import datetime  
chart = st.line\_chart()  
if st.button('Send Report'):  
send\_Email()  
for i in range(len(random\_values)):  
# Get the current time  
current\_time = datetime.now()  
# Write the current time to the CSV file  
with open('times.csv', mode='a') as csv\_file:  
csv\_writer = csv.writer(csv\_file)  
csv\_writer.writerow([current\_time.strftime('%Y-%m-%d %H:%M:%S'),random\_values[i]])  
chart.add\_rows([random\_values[i]])  
time.sleep(5)