PROJECT REPORT

ON

DNS Tunneling Attack Detection using Machine Learning Algorithm

Carried Out at

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

ELECTRONIC CITY, BANGALORE

UNDER THE SUPERVISION OF

Mr. Gopinath P.

Submitted By

Shivprasad Somnath Bulbule (220950125088)

Shashank Arya (220950125081)

Sagar Karbhari Jatode (220950125073)

Rakesh Prasad Yadav (220950125072)

Shubham Arjun Shelke (220950125084)

PG DIPLOMA IN BIG DATA ANALYTICS

C-DAC, BANGALORE

Candidate’s Declaration

We hereby certify that the work being presented in the report entitled **DNS Tunneling Attack DNS Tunneling Attack Detection using Machine Learning Algorithm**, in partial fulfillment of the requirements for the award of PG Diploma Certificate and submitted in the department of PG-DBDA of the C-DAC Bangalore, is an authentic record of our work carried out during the period, 15st September 2022 to 15th March 2023 under the supervision of **Mr. Gopinath P**, C-DAC Bangalore. The matter presented in the report has not been submitted by us for the award of any degree of this or any other Institute/University.

(Name and Signature of Candidate)

Shivprasad Somnath Bulbule (220950125088)

Shashank Arya (220950125081)

Sagar Karbhari Jatode (220950125073)

Rakesh Prasad Yadav (220950125072)

Shubham Arjun Shelke (220950125084)

Counter Signed by

ACKNOWLEDGMENT

We take this opportunity to express our gratitude to all those people who have been directly and indirectly with us during the competition of this project.

We pay thanks to Mr. Gopinath P who has given guidance and a light to us during this major project. His versatile knowledge about “title name “has eased us in the critical times during the span of this Final Project.

We would like to express our sincere thanks to Ms. Snehal Parbat for her valuable guidance and support in completing this project.

We acknowledge here out debt to those who contributed significantly to one or more steps. We take full responsibility for any remaining sins of omission and commission.

Students Name

<u>CERTIFICATE</u>

This is to certify that the work titled **DNS Tunneling Attack Detection using Machine Learning Algorithm** is carried out by Shivprasad Somnath Bulbule (220950125088), Shashank Arya (220950125081), Sagar Karbhari Jatode (220950125073), Rakesh Prasad Yadav (220950125072), Shubham Arjun Shelke (220950125084) the bonafide students of Diploma in Big Data Analytics of Centre for Development of Advanced Computing, Electronic City, Bangalore from 15th September 2022 – 15th March 2023. The Course End Project work is carried out under my direct supervision and completed.

Mr. Gopinath P

C-DAC #68, Electronic City,

Bangalore - 560100, India

ABSTRACT

Domain Name System (DNS) is one of the most important services keeping the Internet communication running. Its main task is to convert domain names to IP addresses and vice versa. DNS tunneling is a method used by malicious users who intend to bypass the firewall to send or receive data and also Attacker can exploit DNS to hide data in DNS tunnel as the organization would not block DNS packets in order to access the Internet. With DNS tunneling, another protocol can be tunneled through DNS. A DNS tunnel can be used for ‘command and control’, data exfiltration or tunneling of any internet protocol (IP) traffic. This has a significant impact on revealing or releasing classified information. Malicious users extract confidential data from compromised computers by encoding the data to be tunneled within a DNS query name, while avoiding firewalls and other network security monitoring mechanisms. We developed machine learning (ML) classifiers for the detection of DNS Tunneling attacks, using a supervised learning. We plan to try out these techniques and improvise on them for better precision in determining tunneling attack.

TABLE OF CONTENT

CHAPTER 1
INTRODUCTION

DNS translates easily memorized domain names to numerical IP addresses which is an essential service related to network and Internet Functionality. For This purpose, DNS protocol uses special message formats and types, like queries and replies. DNS and communicate on port 53 using usually UDP and TCP when the request is larger than 512 octets. RFC 1035.

A DNS server can be authoritative –holding the DNS information -for one zone (example: domain.com) or it can be a local DNS cache serving client DNS queries. DNS queries are of two types: (i) Recursive: recursion is when a DNS server query other DNS server on behalf of original DNS client for name resolution; (ii) Iterative: Forwarded to authoritative servers starting with ROOT servers. Each server refers the client to the next server in the chain, until the current server can fully resolve the request. So, the resolution of www.example domain.com would query a global root server, then the top-level domain “com” server and finally the “exampledomain.com” server.

DNS traffic is often allowed without being inspected by network security devices and almost ignored in network security policies, whichmakes DNS a prone for attacks and misuse. DNS includes some flexible fields used by attackers like TXT record and others.

In 1998, Data transfer over DNS protocol has been discovered and was originally designed as a simple way to bypass the captive portals at the network edge and gain free Wi-Fi access restricted access sites. Currently, transferring data over DNS poses a serious security risk to all organizations.

Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for building mathematical models and making predictions using historical data or information. Currently, it is being used for various tasks such as image recognition, speech recognition, email filtering, Facebook auto-tagging, recommender system, and many more.

CHAPTER 2

LITERATURE SURVEY

Sanjay, Balaji Rajendran et al (2020) DNS Amplification & DNS Tunneling Attacks Simulation, Detection and Mitigation Approaches the technicalities behind DNS amplification and DNS tunneling attacks and presents a number of countermeasures and mitigation techniques to protect against these attacks and the DNS Infrastructure

Mahmoud Sammour DNS Tunneling the DNS tunneling which is an attack that exploit the domain name protocol in order to bypass security gateways. This would lead to lose critical information which is a disastrous situation for many organizations. Recently, researchers have paid more attention in the machine learning techniques regarding the process of DNS tunneling

Farnham, G, Atlasis A et al (2013). Detecting DNS tunneling, SANS Institute InfoSec. This paper is about DNS protocol which enables applications such as web browsers to function based on domain names. They provide a covert channel for malicious activities which represent a significant threat to organizations. These threats can be mitigated using payload analysis and traffic analysis detection techniques.

Caleb Baker et al (2019) Challenges in Effective DNS Query Monitoring. This paper tell about Capturing the contents of DNS queries and analyzing the logged data is a recommended practice for gaining insight into activity on a network and monitoring for unusual behavior. It is likely to make effective deployment of DNS query monitoring solutions significantly more complicated moving forward is unlikely to be the last malware that utilizes it as a method to obfuscate communication. The blocking known public DNS over HTTPS resolvers on enterprise networks seems to be the only effective counter.

Issa kalil et al (2018) A Survey on Malicious Domains Detection through DNS Data Analysis. This paper tell about Malicious domains are one of the major resources required for adversaries to run attacks over the Internet. Due to the important role of the Domain Name System (DNS), extensive research has been conducted to identify malicious domains based on their unique behavior reflected in different phases of the life cycle of DNS queries and responses.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATION

3.1 Product Perspective

A large number of DNS Tunneling utilities exist with a wide range of capabilities. They provide a covert channel for malicious activities which represent a significant threat to organizations. These threats can be mitigated using payload analysis and traffic analysis detection techniques. DNS Tunneling attack detection techniques implemented organizations can reduce the risk associated with DNS tunneling. We plan to identify malicious attack using machine learning algorithm.

3.1.1 Product Functions

Function ID		
F1	**Function**	Training machine learning algorithm
	description	Training the algorithm using the data in csv files
F2	**Function**	Detection of DNS Tunneling using machine learning algorithm.
	description	Detecting malicious and benign DNS request using LightGBM machine learning algorithm

CHAPTER 4
DNS ARCHITECTURE

4.1 DNS name hierarchy

The domain structure of DNS is a tree hierarchy consisting of nodes, areas, domains, subdomains, and other elements. The top of the domain structure is the root zone. Set-ting the root zone is located on multiple servers/mirrors placed around the world. This contains information about all servers in the root zone, and is also responsible for the top level domains (ru, net, org, etc.).The servers of the root zone process and respond to requests, giving information only about the top level domains. The root and top level domains are administered by The Internet Corporation for Assigned Names and Numbers (ICANN). Below the top level domains (TLD), the administration of namespace is delegated to organizations.

FIGURE 1 shows the DNS hierarchy tree. Each node in the tree represents a DNS name. This name can include DNS domains, computers or services. DNS domains can contain both hosts (computers or services) and other domains (subdomains). Each organization is assigned authority for a part of the domain namespace and is responsible for administering DNS domains and computers within this sector of the namespace

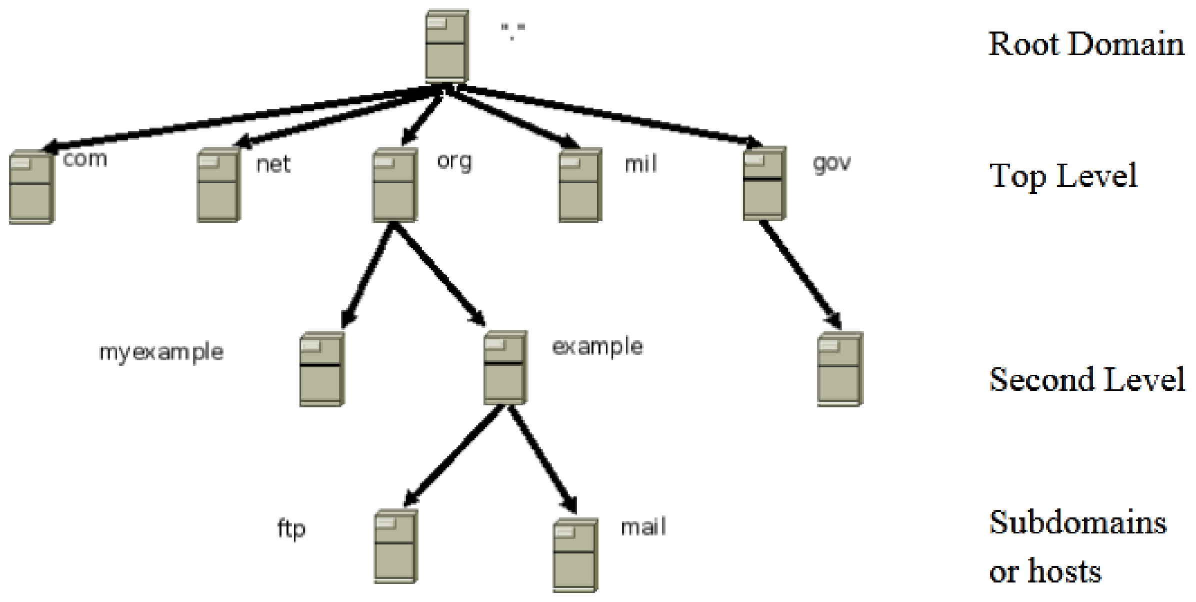


Figure 1 DNS Hierarchy

Zone is any part of the tree of the domain name that is placed as a whole on a DNS server. Zone can be called a "zone of responsibility". The purpose of the allocation of the tree in separate areas the transfer of responsibility to persons or organizations. Each zone has at least one authoritative server DNS which stores all the information about the zone for which it is responsible.

Domain is a named branch or subtree in the tree of DNS structure, therefore, it is a particular node, including all subordinate nodes. Each node in the DNS hierarchy is separated from its parent by a dot. The domain name begins with a dot (the root do-main) and passes through the domains of the first, second, and third (if necessary) levels and ends with hostname. Thus, domain name fully reflects the structure of DNS hierarchy. Often, the last dot (the designation of the root domain) in the domain name is omitted.

Fully Qualified Domain Name (FQDN) domain name that uniquely identifies the domain name and includes the names of all parent domains in the DNS hierarchy, including the root. It is like an analogue of the absolute path in file system. A Root Domain Top Level Second Level Sub domains or hosts 10 example of this is present in *Figure 2* that introduces the domain name ftp.example.org from *Figure 1*.

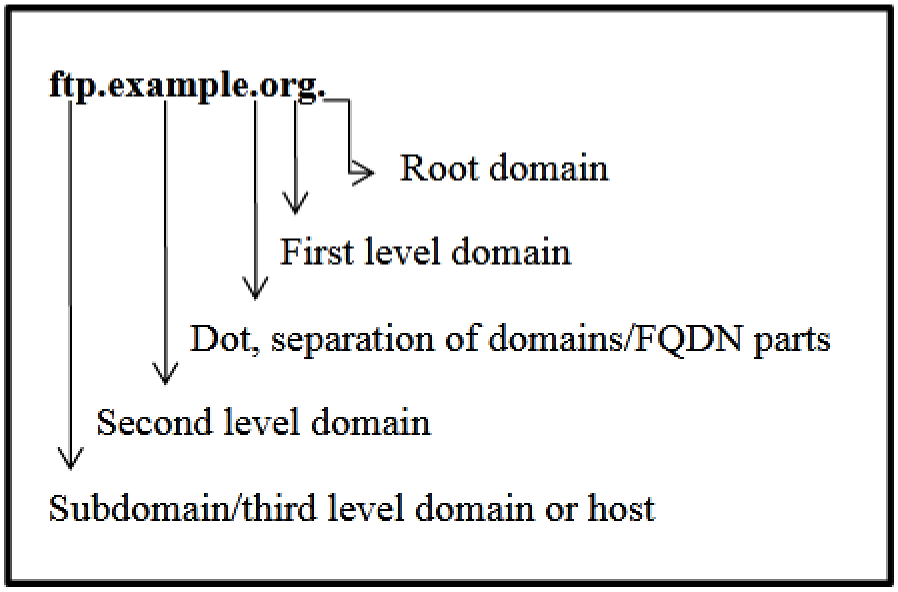

ftp.example.org.
Root domain
First level domain
Dot, separation of domains/FQDN parts
Second level domain
Subdomain/third level domain or host

Figure 2. Fully Qualified Domain Name example.

4.2 Log Analysis of Bind

Real time log analysis visualize data being provided by logs and helps the administrator to understand what is going on based on the statistics of the data.

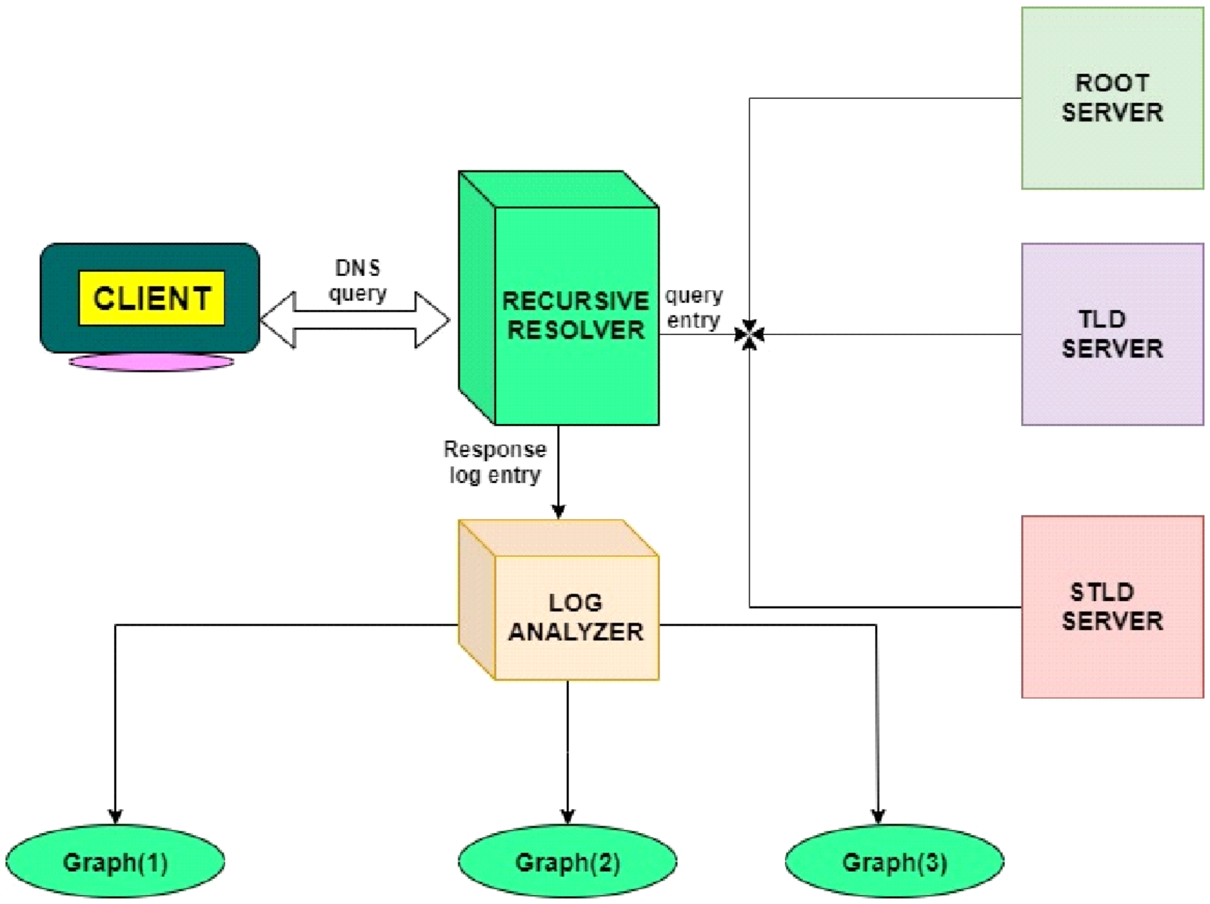


Figure 3 DNS Log Analysis

4.3 DNS logging

Basically, statistics of 1DNS is done by log file which maintains a history of page requests. More recent entries are typically appended to the end of the file. Information about the request, including client IP address, request date/time, page requested, response time, response nature, query type, etc.

4.4 DNS monitoring

DNS monitoring can be used for logging of all communications between clients and DNS servers**.** Regarding the DNS platform and number of DNS servers, DNS log is the single point where you have all the queries and responses in the form of flow records. Flow records can be filtered using all extracted DNS parameter enabling you to tackle problems very quickly**.**

4.5 Why Analysis?

Malicious domains are key components to a variety of cyber-attacks. Several recent techniques are proposed to identify malicious domains through analysis of DNS data. The general approach is to build classifiers based on DNS-related local domain features. One potential problem is that many local features, e.g., domain name patterns and temporal patterns, tend to be not robust. Attackers could easily alter these features to evade detection without affecting much their attack capabilities. We take a complementary approach. Instead of focusing on local features, we propose to discover and analyses global associations among domains. The key challenges are (1) to build meaningful associations among domains; and (2) to use these associations to reason about the potential maliciousness of domains. To avoid detection, malicious domains exhibit dynamic behavior by, for example, frequently changing the malicious domain-IP resolutions and creating new domains. This makes it very likely for attackers to reuse resources. It is indeed commonly observed that over a period of time multiple malicious domains are hosted on the same IPs and multiple IPs host the same malicious domains, which creates intrinsic association among them. For the second challenge, we develop a graph-based inference technique over associated domains. Our approach is based on the intuition that a domain having strong associations with known malicious domains is likely to be malicious. Carefully established associations enable the discovery of a large set of new malicious domains using a very small set of previously known malicious ones. Our experiments over a public passive DNS database show that the proposed technique can achieve high true positive rates (over 95%) while maintaining low false positive rates (less than 0.5%). Further, even with a small set of known malicious domains, our technique can discover a large set of potential malicious domains.

CHAPTER 5
SYSTEM DESIGN

The Domain name server (DNS) is a distributed dynamic database that contains in-formation about the hosts in the Internet. Distributed means that each server does not contain information about all of the Internet namespace. Each server is responsible for its sectors of the space. Most commonly, it includes the machine name, IP address and data for mail routing.

This system was introduced for the users' convenience and the main task is the con-version of human-readable computer names into IP addresses and vice versa. There-fore, in a case when a computer has a lack of information about certain device, it sends a request to the DNS server to obtain the necessary data.

5.1 How does DNS work?

There are two ways to resolve names. The first is recursive -a method in which the DNS servers do all work to give final data back to the host. The Second is an iterative query, the server will give back a referral to other DNS servers which might have the answer.

The recursive server returns only real answers and error messages. It keeps track of references are exempt from this duty of the client. The basic procedure for the resolution of the request, in essence, is the same. The only difference is that the name server takes care of the processing of references, not passing them back to the client.

This method works as follows:

1. The resolver of the client sends a recursive query to a DNS server on some Server1.
2. Server1 has no needed data (it is not authoritative for the required zone or there are no records about it in the cache) and sends a query to the Root.
3. Since the data cannot be resolved (root works only as an iterative server), the root server returns to the Server IP address of the DNS server which is responsible for the .com zone.

4. Server1 sends a query to the server responsible for the .com zone.
5. Since data cannot be resolved, the server returns to the Server1 IP address of the DNS server responsible for the area example.com.
6. Server1 Send An iterative query to the server responsible for the example.Com area.
7. Server1 gets the desired data (for example, mail.example.com).
8. Server sends the data back to the client’s resolver.

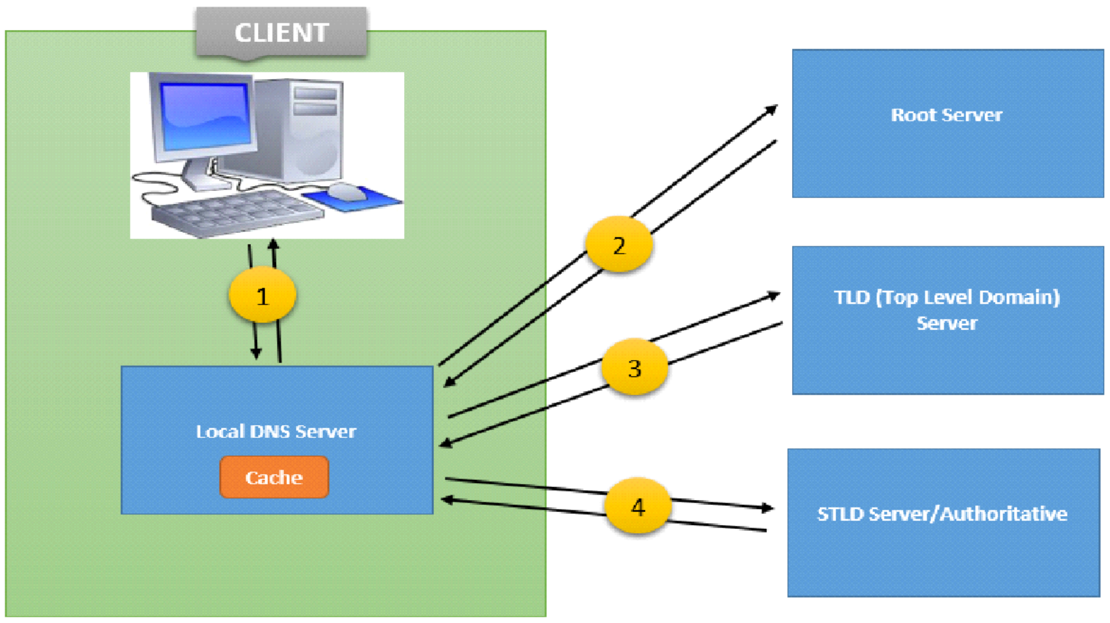


Figure 4 Working Of DNS

5.2 DNS Tunneling

Tunneling allows transmission of data using a certain infrastructure encoding data of other programs and protocols in DNS queries and responses without alerting any firewalls intrusion detection system. The original intention was to bypass captive portals in Wi-Fi Hotspots at airports or hotels to acquire free internet access. DNS tunneling is a client-server model requiring a client to be compromised through malware, phishing or social engineering with the only requirement of access to internal. DNS server. At the infected DNS client level, a persistent backdoor with a DNS Tunnel will thus be established. DNS tunnels can be used to exfiltrate important and confidential data from any organizations network (Data Exfiltration) in the form of Command and control channel. Communication channel between the target host and the command and control server. It embeds data and commands in DNS queries and responses. Also, it includes full remote access of the compromised host.

5.3 Working of DNS Tunneling

DNS tunneling requires a compromised client system to have external network connectivity and a Rogue DNS server controlled by the malicious user that can act as an authoritative server to execute the server-side tunneling and data payload executable programs. After being infected by malware, the DNS client starts issuing recursive DNS queries addressed to a domain name controlled by the threat actor. The local DNS server then forwards the queries iteratively to authoritative servers which should appear as normal to the local firewall. As shown in Figure 1.

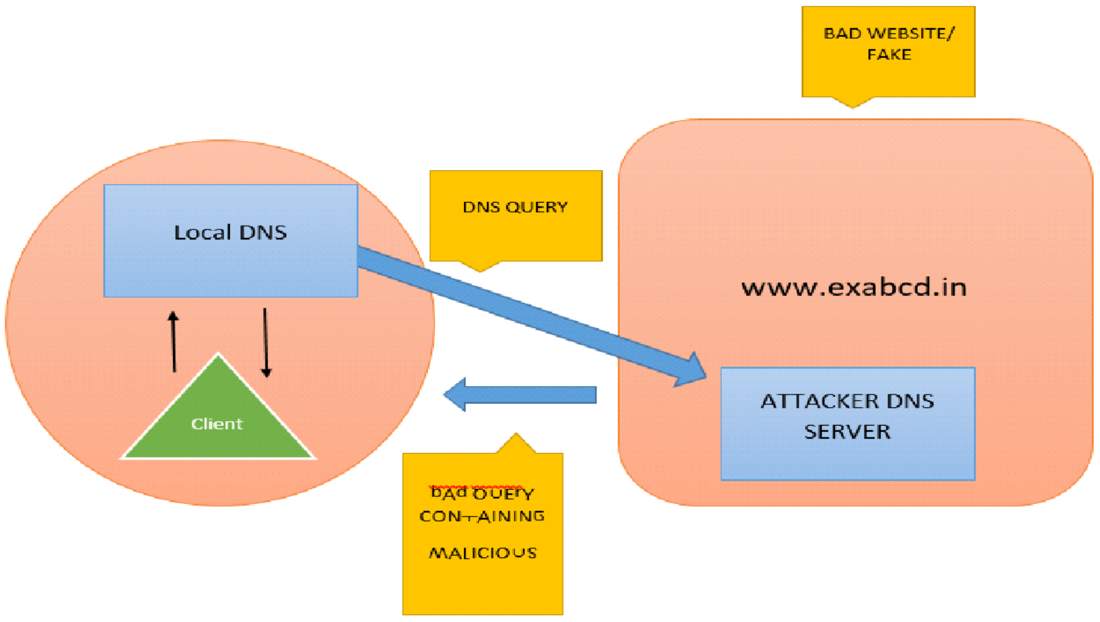


Figure 5 Working Of DNS Tunneling

5.4 Functionality DNS Log

DNS Analytics allow you to see your domains' query activity as raw data logs or in visual forms such as: line and bar charts, interactive maps, and filterable tables.

- Quickly identify traffic anomalies
- Gather insight into your DNS infrastructure
- Pinpoint system misconfigurations
- Find stale/unused records
- Compare usage trends over time

5.5 Important terms in DNS Log

The DNS log is a file, usually in . txt or .csv format, that contains highly detailed data on all DNS information sent and received by the DNS server. The important factors are Source IP, Destination IP, Source Port, Destination Port, Time Stamp, Duration, Flow Bytes Sent, Flow Sent Rate, Flow Bytes Received, Flow Received Rate, Packet Length Variance, Packet Length Standard Deviation, Packet Length Mean, Packet Length Median, Packet Length Mode, Packet Length Skew From Median, Packet Length Skew From Mode, Packet Length Coefficient of Variation, Packet Time Variance, Packet Time Standard Deviation, Packet Time Mean, Packet Time Median, Packet Time Mode, Packet Time Skew From Median, Packet Time Skew From Mode, Packet Time Coefficient of Variation, Response Time, Time Variance, Response Time, Time Standard Deviation, Response Time ,Time Mean, Response Time, Time Median, Response Time, Time Mode, Response Time ,Time Skew From Median, Response Time, Time Skew From Mode, Response Time, Time Coefficient of Variation, DoH, Using these attributes the patterns in DNS log can be found.

CHAPTER 6

Implementation of Machine learning Algorithm

Machine learning tools are algorithmic applications of artificial intelligence that give systems the ability to learn and improve without ample human input. There are many tools used in machine learning.

6.1 Preprocessing of machine Learning Algorithm

Importing important libraries-

A module is a file with python code. The code can be in the form of variables, functions, or class defined. Modules used in this project are pandas, numpy, matlotlib and seaborn.

```
# importing all libraries


import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt


import warnings

warnings.filterwarnings('ignore')

import ipaddress

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import RobustScaler


from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score , classification_report ,
confusion_matrix
```

```
from sklearn.model_selection import GridSearchCV
import lightgbm as lgb


from imblearn.over_sampling import ADASYN , RandomOverSampler , SMOTE
```

Uploading .csv file for exploratory data analysis

```
df = pd.read_csv('all.csv')
```

Datatype of features

```
df.dtypes
```

Output exceeds the size limit. Open the full output data in a text editor

```
Unnamed: 0 int64
SourceIP object
DestinationIP object
SourcePort int64
DestinationPort int64
TimeStamp object
Duration float64
FlowBytesSent int64
FlowSentRate float64
FlowBytesReceived int64
FlowReceivedRate float64
PacketLengthVariance float64
PacketLengthStandardDeviation float64
PacketLengthMean float64
PacketLengthMedian float64
```

PacketLengthMode int64

PacketLengthSkewFromMedian float64

PacketLengthSkewFromMode float64

PacketLengthCoefficientofVariation float64

PacketTimeVariance float64

PacketTimeStandardDeviation float64

PacketTimeMean float64

PacketTimeMedian float64

PacketTimeMode float64

PacketTimeSkewFromMedian float64

ResponseTimeTimeSkewFromMedian float64

ResponseTimeTimeSkewFromMode float64

ResponseTimeTimeCoefficientofVariation float64

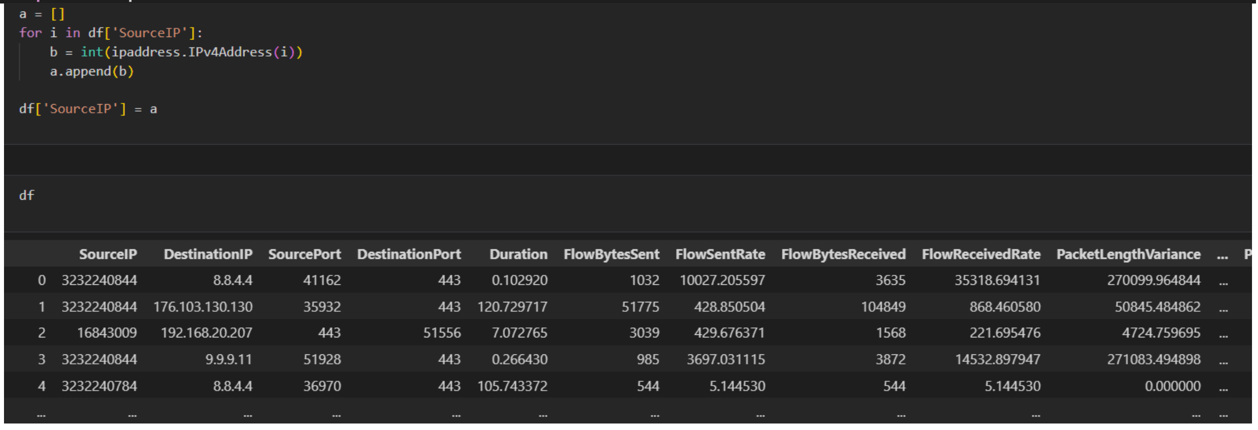
Label object

Dropping unnecessary attributes

```
df.drop(['Unnamed: 0','TimeStamp'] , axis=1 , inplace=True)
```

Converting IP adresses to integer

```
import ipaddress
```



```
a = []
for i in df['SourceIP']:
    b = int(ipaddress.IPv4Address(i))
    a.append(b)

df['SourceIP'] = a
```

```
df
```

```
       SourceIP  DestinationIP  SourcePort  DestinationPort    Duration  FlowBytesSent   FlowSentRate  FlowBytesReceived  FlowReceivedRate  PacketLengthVariance  ...  P
0    3232240844        8.8.4.4       41162              443    0.102920           1032   10027.205597               3635      35318.694131         270099.964844  ...
1    3232240844 176.103.130.130      35932              443  120.729717          51775     428.850504             104849        868.460580          50845.484862  ...
2      16843009 192.168.20.207         443            51556    7.072765           3039     429.676371               1568        221.695476           4724.759695  ...
3    3232240844       9.9.9.11       51928              443    0.266430            985    3697.031115               3872      14532.897947         271083.494898  ...
4    3232240784        8.8.4.4       36970              443  105.743372            544       5.144530                544          5.144530              0.000000  ...
...         ...            ...         ...              ...         ...            ...            ...                ...               ...                   ...  ...
```

Handling null values using mean of that column

```
df['ResponseTimeTimeMedian'] =
df['ResponseTimeTimeMedian'].fillna(df['ResponseTimeTimeMedian'].mean())


df['ResponseTimeTimeSkewFromMedian'] =
df['ResponseTimeTimeSkewFromMedian'].fillna(df['ResponseTimeTimeSkewFromMedian'].
mean())
```

Converting Target variable into integer using Label encoder

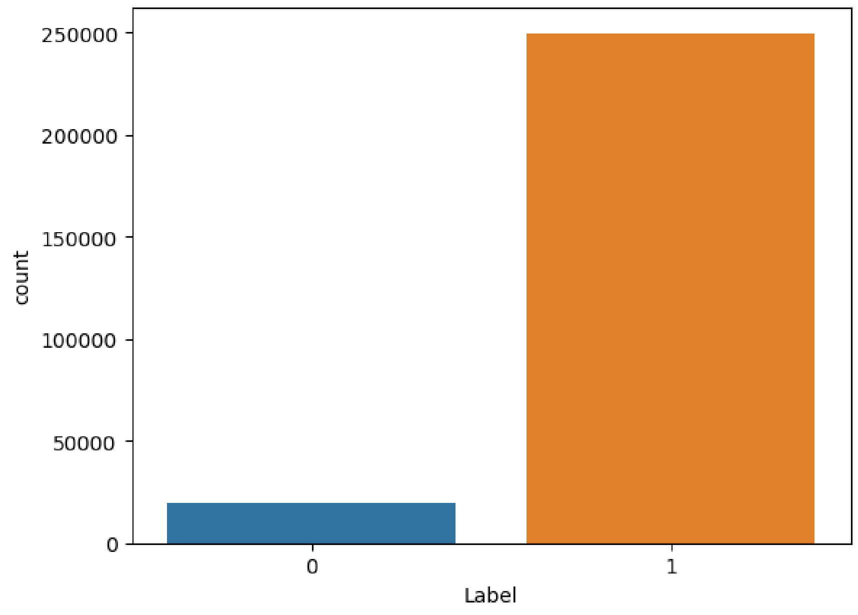
```
from sklearn.preprocessing import LabelEncoder
en = LabelEncoder()
df['Label'] = en.fit_transform(df['Label'])
df['Label']
df['Label'].unique()
df['Label'].value_counts()
```

Separating independent and dependent(target) variable

```
x = df.drop('Label' , axis=1)
y = df.Label
x
y
```

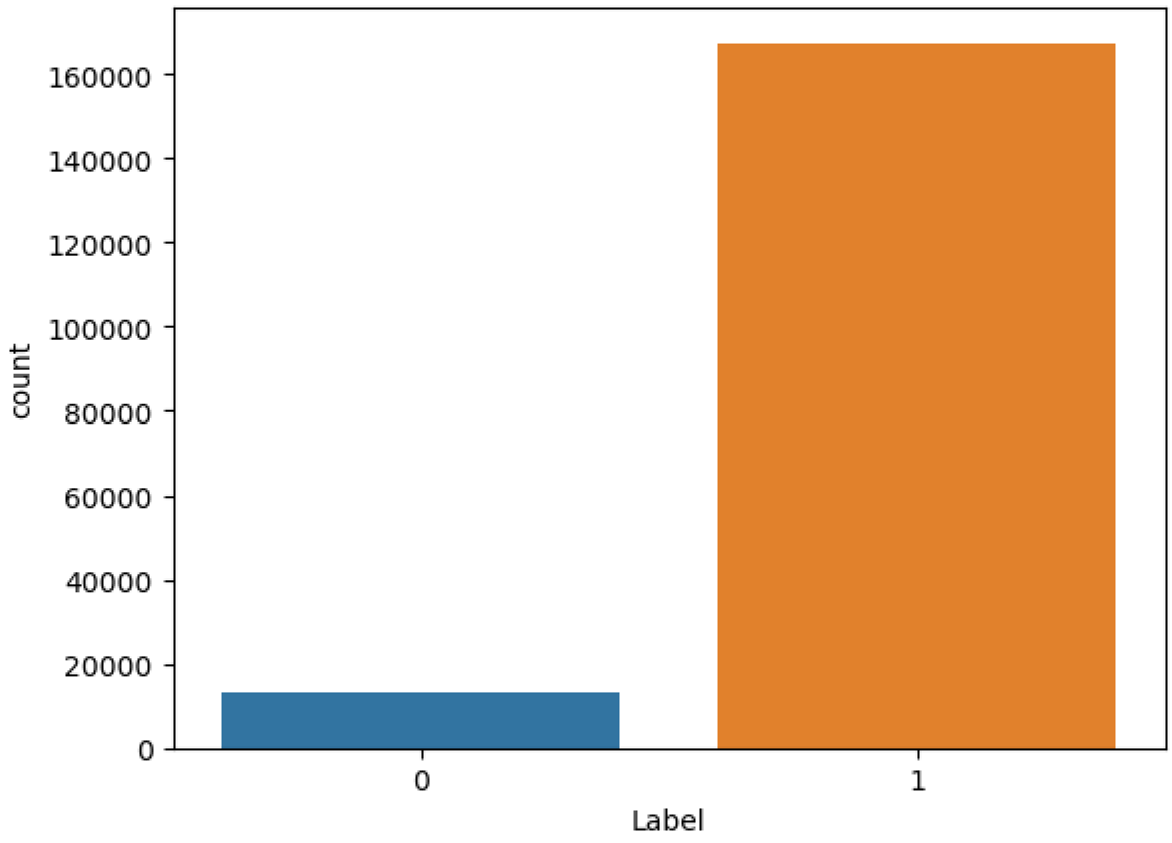
Checking data is balanced or not

```
sns.countplot(y)
```



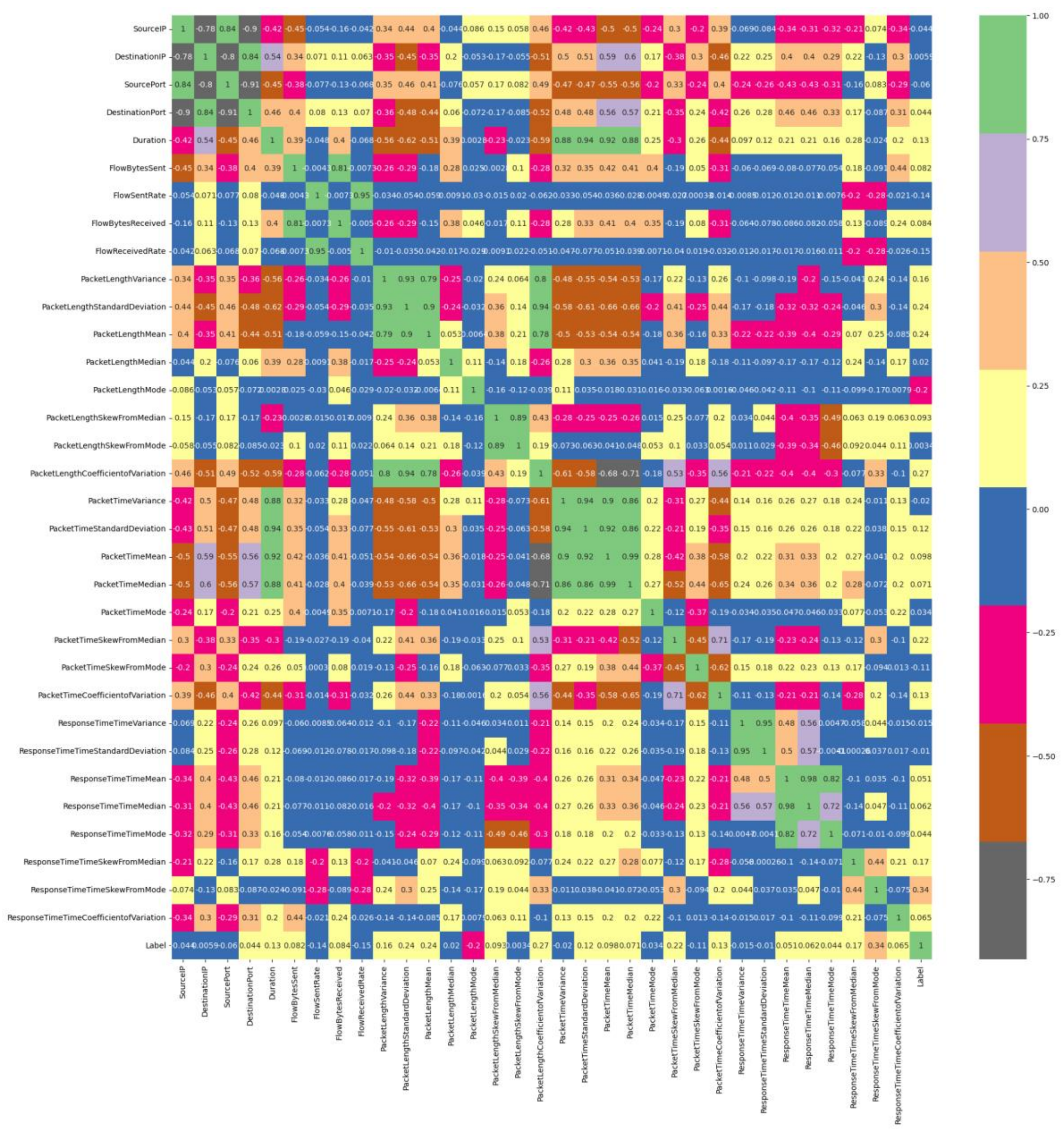
6.2 Splitting data into training and testing

```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(x,y,test_size=0.33 ,
random_state=42)
x_train.shape , x_test.shape , y_train.shape , y_test.shape
x_train
x_test
y_train
y_test
sns.countplot(y_train)
plt.show()
```



Correlation matrix

```
corr = x_train.corr()
#plotting coorelation using heatmap
corr = df.corr()

fg = plt.figure(figsize=(22,22))

sns.heatmap(corr , annot=True , cmap=plt.cm.Accent_r)
```



Creating function to get highly intercorrelated features

```
def correlation(dataset , threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)

    return col_corr
```

Removing highly intercorrelated features

```
x_train.drop(corr_features , axis=1 , inplace=True)

x_test.drop(corr_features , axis=1 , inplace=True)
x_train
x_test
x_train.shape , x_test.shape
```

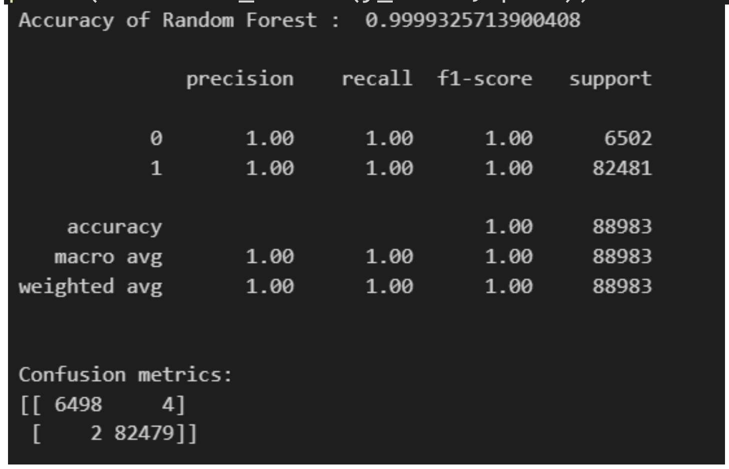
Scaling the features using Robust scaler

```
from sklearn.preprocessing import RobustScaler
rob = RobustScaler()
x_train_scaled = rob.fit_transform(x_train)
x_train_scaled
x_test_scaled = rob.fit_transform(x_test)
x_test_scaled
sns.countplot(y_train)
plt.show()
```

6.3 Applying classification algorithms without balancing data

1)Random Forest Classifier-

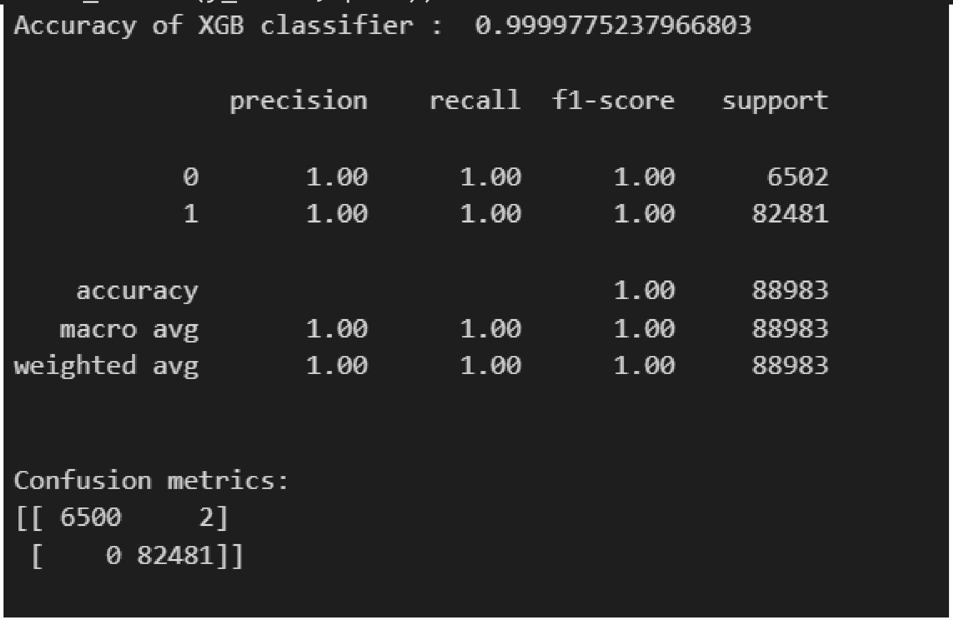
```
rf = RandomForestClassifier()
rf.fit(x_train_scaled , y_train)
pred = rf.predict(x_test_scaled)
print('Accuracy of Random Forest : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```



```
Accuracy of Random Forest :  0.9999325713900408

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6502
           1       1.00      1.00      1.00     82481

    accuracy                           1.00     88983
   macro avg       1.00      1.00      1.00     88983
weighted avg       1.00      1.00      1.00     88983


Confusion metrics:
[[ 6498     4]
 [    2 82479]]
```

1)XGB classifier

```
xgb = XGBClassifier()
xgb.fit(x_train_scaled , y_train)
pred = xgb.predict(x_test_scaled)
print('Accuracy of XGB classifier : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```



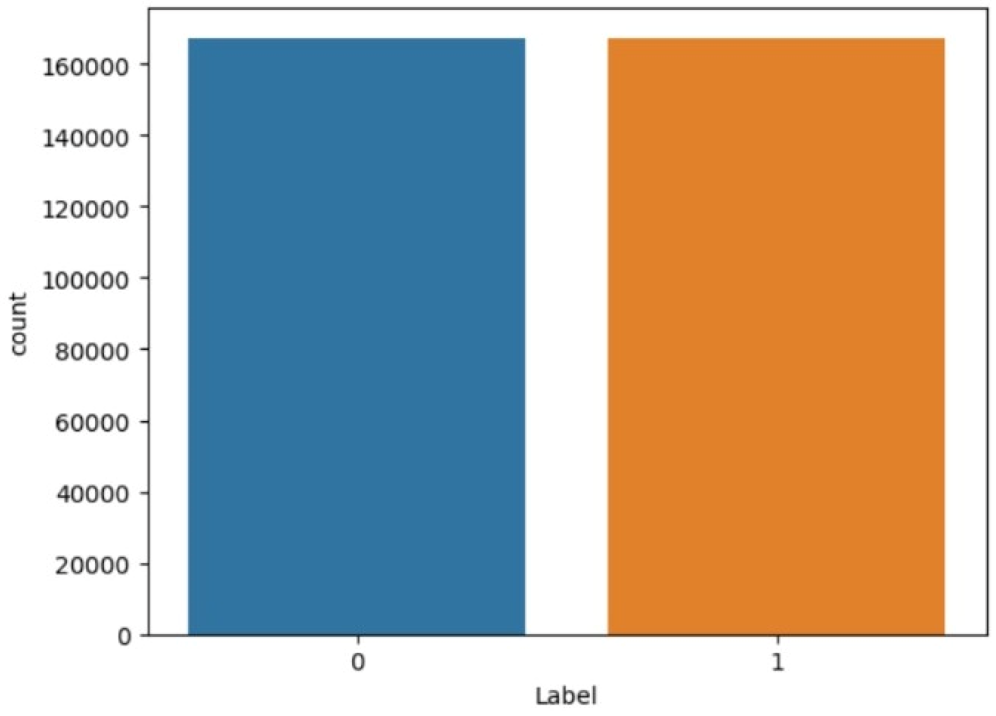
```
Accuracy of XGB classifier :  0.9999775237966803

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6502
           1       1.00      1.00      1.00     82481

    accuracy                           1.00     88983
   macro avg       1.00      1.00      1.00     88983
weighted avg       1.00      1.00      1.00     88983


Confusion metrics:
[[ 6500     2]
 [    0 82481]]
```

6.4 Applying classification algorithms after balancing data

Balancing data using Over-sampling

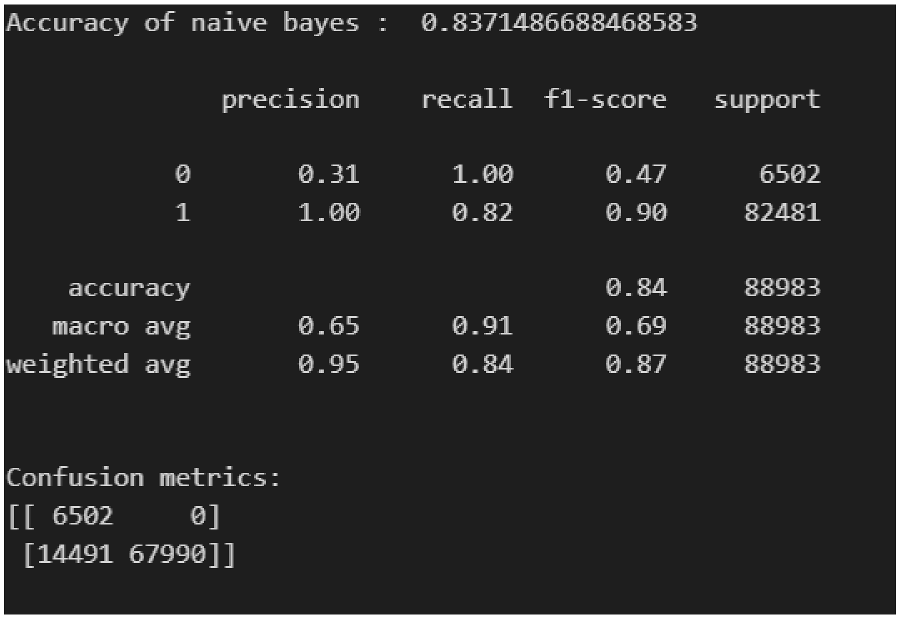
```
from imblearn.under_sampling import NearMiss
ad = ADASYN()
x_train , y_train = ad.fit_resample(x_train_scaled , y_train)
rf = RandomForestClassifier(n_estimators=20, max_depth=8 , max_features=0.6 ,
max_samples=0.75)
rf.fit(x_train , y_train)
pred = rf.predict(x_test_scaled)
print('Accuracy of naive bayes : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```

```
[52]: ad = ADASYN()

      x_train , y_train = ad.fit_resample(x_train_scaled , y_train)

[53]: sns.countplot(y_train)

[53]: <AxesSubplot:xlabel='Label', ylabel='count'>
```



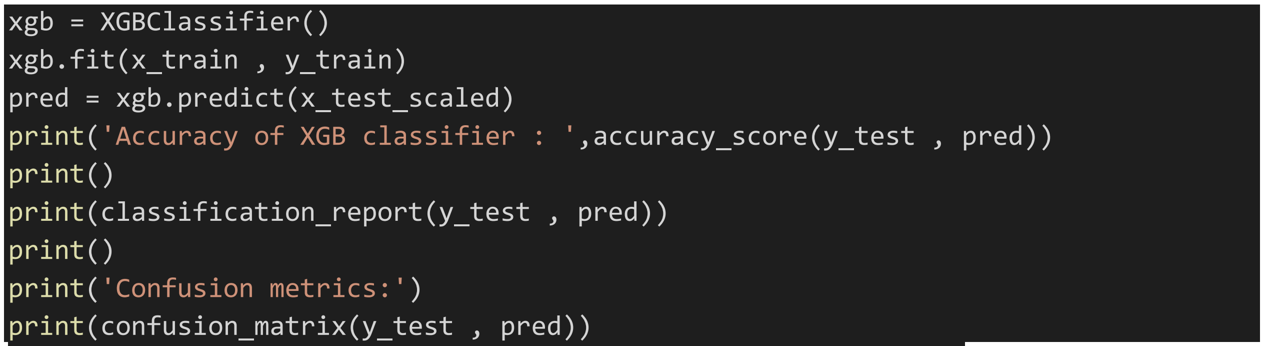
1)Random Forest-

```
rf = RandomForestClassifier()
rf.fit(x_train , y_train)
pred = rf.predict(x_test_scaled)
print('Accuracy of naive bayes : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```

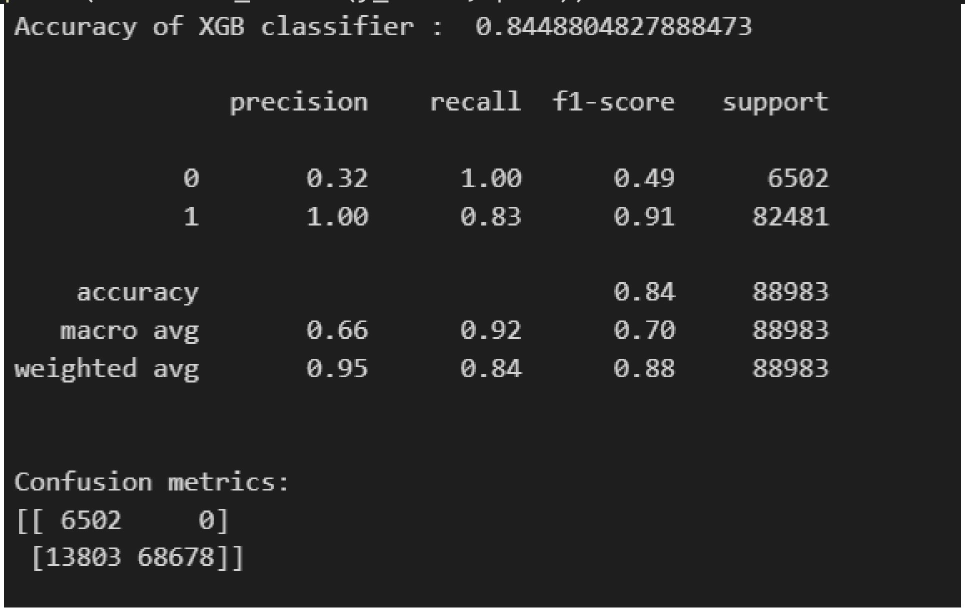


```
Accuracy of naive bayes :  0.8371486688468583

              precision    recall  f1-score   support

           0       0.31      1.00      0.47      6502
           1       1.00      0.82      0.90     82481

    accuracy                           0.84     88983
   macro avg       0.65      0.91      0.69     88983
weighted avg       0.95      0.84      0.87     88983


Confusion metrics:
[[ 6502     0]
 [14491 67990]]
```

2)XGB Classifier-



```
xgb = XGBClassifier()
xgb.fit(x_train , y_train)
pred = xgb.predict(x_test_scaled)
print('Accuracy of XGB classifier : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```



```
Accuracy of XGB classifier :  0.8448804827888473

              precision    recall  f1-score   support

           0       0.32      1.00      0.49      6502
           1       1.00      0.83      0.91     82481

    accuracy                           0.84     88983
   macro avg       0.66      0.92      0.70     88983
weighted avg       0.95      0.84      0.88     88983


Confusion metrics:
[[ 6502     0]
 [13803 68678]]
```

```
As per above results, we can conclude that RF and XGB giving good accuracy and
performance. Now we can perform Hyperparameter Tuning to Improve performance of
those models
```

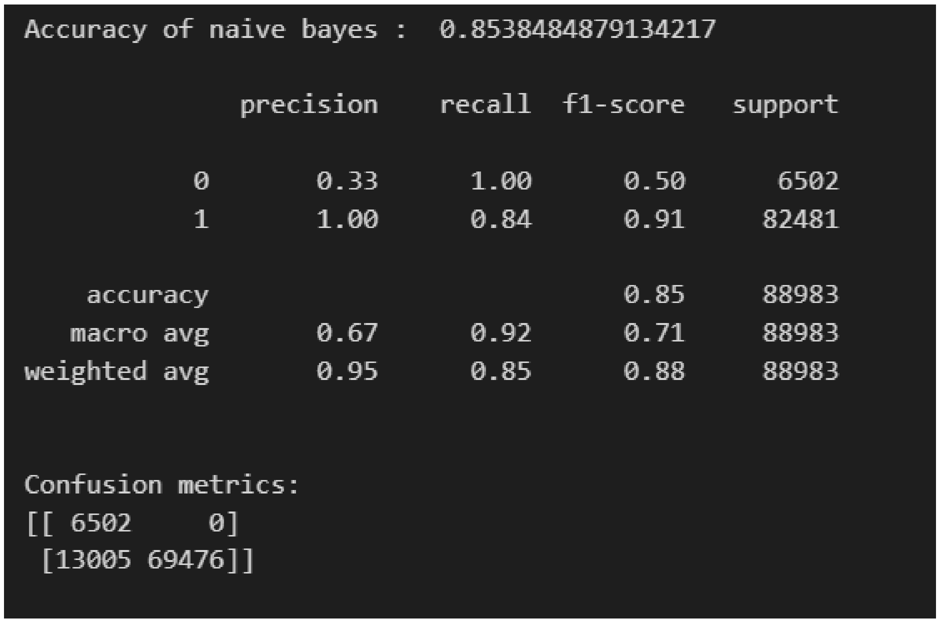
6.5 Hypertuning the models

1)Random Forest with Hyperparameter tuning

```
rf = RandomForestClassifier()
n_estimators = [20,60,100,120]
max_features = [0.2,0.6,1.0]
max_depth = [2,8,None]
max_samples = [0.5,0.75,1.0]
param_grid = {'n_estimators' : n_estimators ,
              'max_features' : max_features ,
              'max_depth'  : max_depth ,
               'max_samples' : max_samples
             }
param_grid
grid = GridSearchCV(estimator = rf ,
                    param_grid = param_grid ,
                    cv= 5,
                    verbose = 1 ,
                    n_jobs = -1)
grid.fit(x_train , y_train)
grid.best_params_
grid.best_score_
0.9999252895031752
```

Accuracy using above parameters

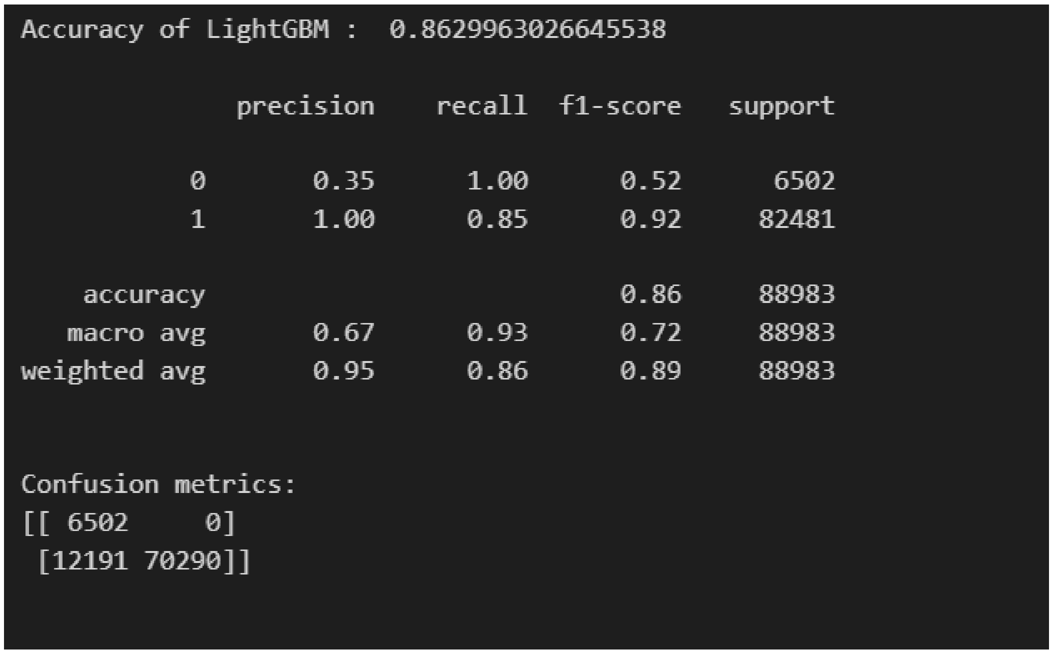
```
rf = RandomForestClassifier(n_estimators=50, max_depth=8 , max_features=0.8 ,
max_samples=0.80)
rf.fit(x_train , y_train)
pred = rf.predict(x_test_scaled)
print('Accuracy of naive bayes : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```



```
Accuracy of naive bayes :  0.8538484879134217

              precision    recall  f1-score   support

           0       0.33      1.00      0.50      6502
           1       1.00      0.84      0.91     82481

    accuracy                           0.85     88983
   macro avg       0.67      0.92      0.71     88983
weighted avg       0.95      0.85      0.88     88983


Confusion metrics:
[[ 6502     0]
 [13005 69476]]
```

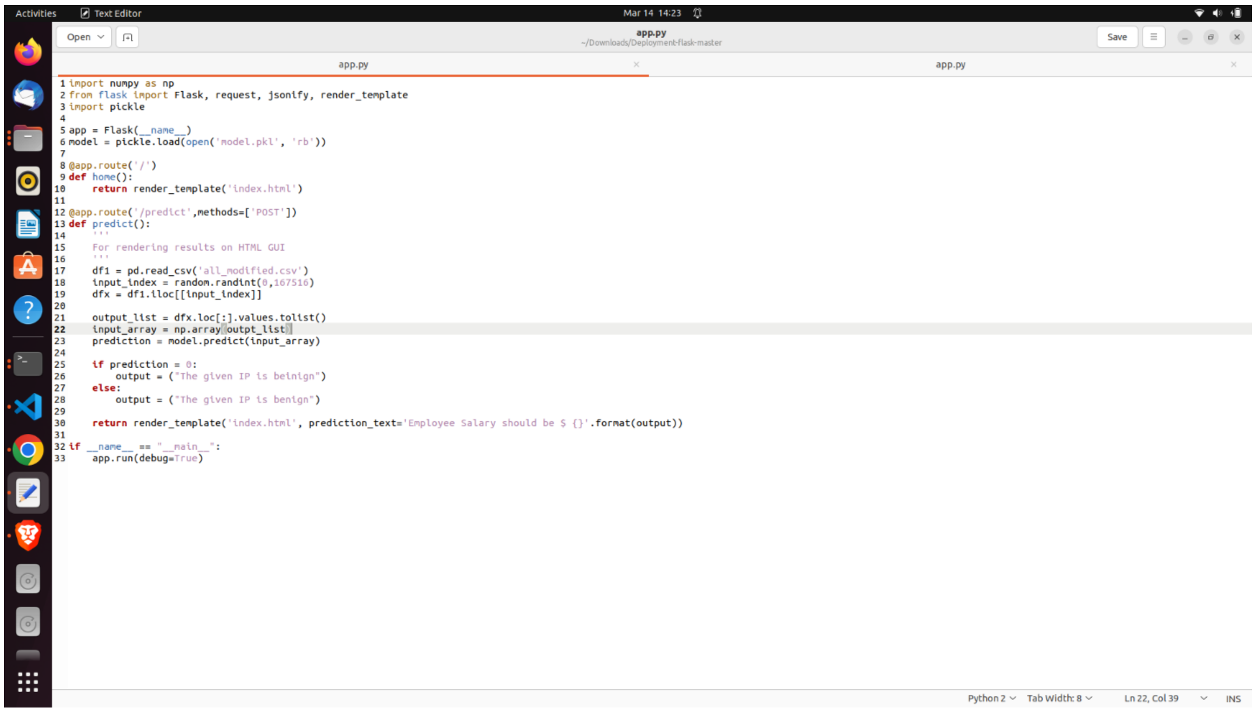
2) LightGBM with Hyperparameter tuning

```
import lightgbm as lgm
clf = lgm.LGBMClassifier()
clf.fit(x_train , y_train)
pred = clf.predict(x_test_scaled)
print('Accuracy of LightGBM : ',accuracy_score(y_test , pred))
print()
print(classification_report(y_test , pred))
print()
print('Confusion metrics:')
print(confusion_matrix(y_test , pred))
```



```
Accuracy of LightGBM :  0.8629963026645538

              precision    recall  f1-score   support

           0       0.35      1.00      0.52      6502
           1       1.00      0.85      0.92     82481

    accuracy                           0.86     88983
   macro avg       0.67      0.93      0.72     88983
weighted avg       0.95      0.86      0.89     88983


Confusion metrics:
[[ 6502     0]
 [12191 70290]]
```

6.6 DNS request prediction using Flask API



```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    df1 = pd.read_csv('all_modified.csv')
    input_index = random.randint(0,167516)
    dfx = df1.iloc[[input_index]]

    output_list = dfx.loc[:].values.tolist()
    input_array = np.array(outpt_list)
    prediction = model.predict(input_array)

    if prediction = 0:
        output = ("The given IP is beinign")
    else:
        output = ("The given IP is benign")

    return render_template('index.html', prediction_text='Employee Salary should be $ {}'.format(output))

if __name__ == "__main__":
    app.run(debug=True)
```

CHAPTER 7

CONCLUSION

A large number of DNS Tunneling utilities exist with a wide range of capabilities. They provide a covert channel for malicious activities which represent a significant threat to organizations. However these threats can be detected as shown some of the methods we used in this paper. While some utilities have been available for several years, others have been recently developed with improved capability.

DNS tunneling represents a significant threat to organizations. Command and control over protocol has full remote access to a compromised endpoint. The other threat is data exfiltration. DNS tunneling provides a covert channel for data exfiltration. Although inefficient for data transfer DNS tunneling can be used to easily exfiltrate high value data such as password hashes or sensitive documents.

Machine learning algorithms like XGBoost, Random Forest has been used to train the model to detect the DNS tunneling. With hyperparameter tuning the performance of these models have been improved.

Chapter 8

REFERENCES

- Ejaz Ahmad and Kashif Sarwar, A Comparative Analysis on Existing DNS Performance Measurement Mechanisms
- Sanjay, Balaji Rajendran et al (2020), DNS Amplification & DNS Tunneling Attacks Simulation, Detection and Mitigation
- Yasir Faraj Mohammed, Network-Based Detection and Prevention System against DNS Based Attacks
- Mohammad Al-Fawa'reh et al, Detecting Malicious DNS Queries Over Encrypted Tunnels Using Statistical Analysis and Bi-Directional Recurrent Neural Networks