SERVER STRESS TEST FOR DDoS ATTACK USING MACHINE LEARNING

ABSTRACT

Attacks on the web servers and web application is the most common form of attacks that are carried out nowadays. The main reason being is most of the web application or services are vulnerable to attacks and can be easily compromised.

One of the popular attacks used is DdoS.

DDoS stand for Distributed denial of services. Most recent websites and web servers are unable to withstand strong attacks like a DDoS attack. These private servers don't have protection against simple attacks and are easily compromised. Distributed Denial of Service (DDoS) attacks are the most damaging attacks that block everyone from using your services. Websites and applications face cyber threats daily by cyber attackers who want to exploit any IT infrastructure vulnerability. Once your website or IT infrastructure is compromised, your business processes and sensitive data face imminent threats that can bring down your organization. It is vital to test your website and applications for protection against DDoS attacks.

INTRODUCTION

DDoS attacks happen every day, and most of the time, it is hard to flag the ill-intended traffic from normal traffic. However, you can be better prepared to counter and have measures in place so that you can defend your website against DDoS attacks.

Common Types of DDoS Attacks

A DoS attack can target any component of your network and IT infrastructure. Attackers look for the opportunity to exploit any vulnerabilities in different layers of your network.
The following are some common DDoS attacks that we see very often:

Application Layer Attacks

These attacks target your network's application layer by sending HTTP traffic load with malicious intent. When an HTTP request comes to the server, to send a response, the server performs multiple tasks such as load files, querying the database, computing the request, preparing the response, etc. With such a huge amount of traffic, the server gets overloaded, and exhausts infrastructure resources and ultimately goes down. Since it is hard to classify these requests as malicious requests due to their nature being similar to actual users, the application layer DDoS attacks are hard to prevent.

Protocol Attacks
These attacks bring down the service by exhausting intermediate resources like state table capacity, load balancers, firewalls, TCP handshakes, etc. For example, attackers can send a TCP handshake request for connection initialization, the server sends back the response and waits for confirmation from the client. But the client never sends the confirmation, and the server keeps waiting for it, causing the server resources to exhaust. These attacks are also called state-exhaustion attacks.

Why Perform DDoS Stress Testing

- Identify and resolve website infrastructure issues and bottlenecks before the DDoS attacks.
- Find out the breaking point for your website under overload conditions and optimize for robustness.
- Planning for an incident response procedure.
- Devising DDoS mitigation and prevention strategies.
- Scaling and securing IT assets for more resilience.
- Evaluating third-party services for DDoS attack scenario.

PROBLEM STATEMENT

Testing the stress on the server during such attacks and improve the security on basis of the level of withstanding of website during such attacks as Distributed Denial of Service (DDoS) attacks are some of the most common, notorious, and damaging cyber attacks. These attacks cause a great deal of money to organizations by bringing down their website and interrupting business processes and consumer service.

LITERATURE SURVEY

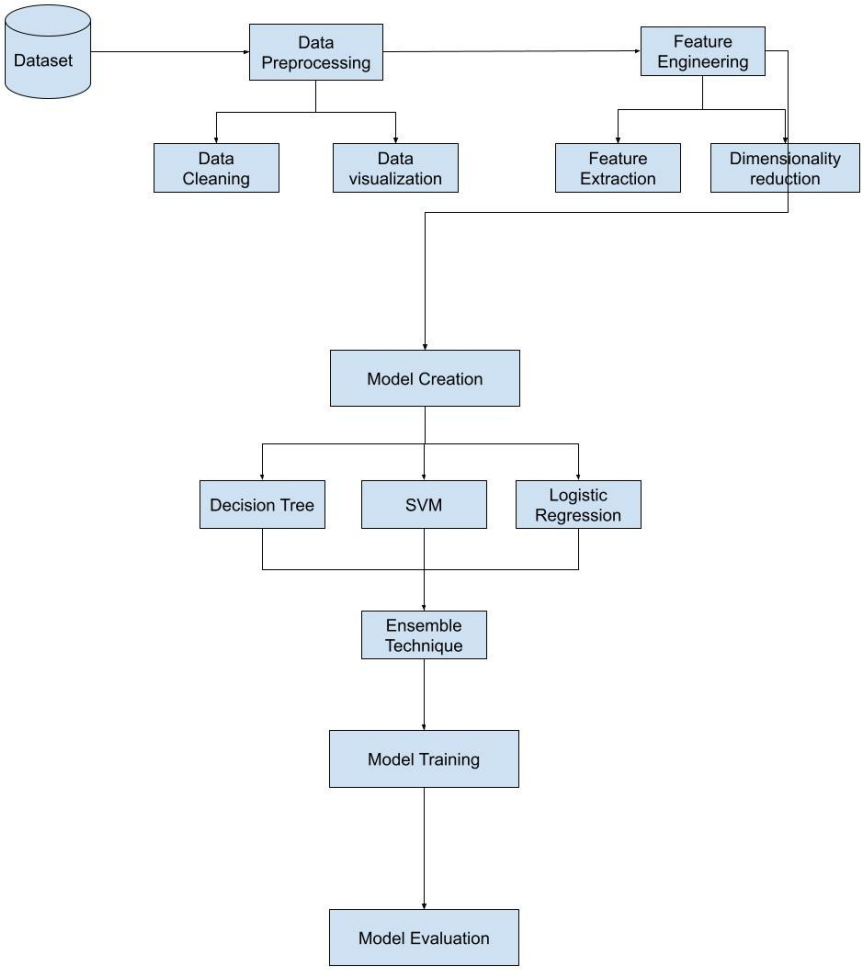
Paper	Algorithm	Advantages	Issues	Metrics used
DDoS Attack Detection Method Based on SVM in Software Defined Network	Support vector machine learning algorithm	By sdn experimental environment the detection rate is high and false alarm is low	Very high time complexity	Accuracy
ML Based Application Layer DdoS Attack Detection Using Artificial Neural Networks	Multilayer Perceptron with genetic machine learning algorithm(MLP-GA), Back propagation algorithm	Correct and precise evaluation metric values and has achieved accuracy of 98%	The prevention mechanism is not illustrated in the paper	Accuracy
Ddos Attack Detection and Mitigation Using ML: Methods, Practices and Solutions	Artificial Neural networks, Self organizing map, Fuzzy logic	i)low performance overhead in switches and routers ii)Not only scales better but also supported by large switch vendors	This Model is not a silver bullet solution to all types of DDoS attacks	i)Entropy ii)Connection rate
Distributed denial of service attack detection using Naive Bayes and KNN	Navie bayes and k nearest neighbour	The detection rate is high in both knn and naive algorithm	Compared to naive bayes algorithm knn performed with more accuracy	Accuracy

DoS/DDoS Attack Detection Using Artificial Neural Networks	Artificail Neural Network	Detected known ddos attacks with full cent and with unknown it detected upto 95%	it cannot handle DDoS attacks that use encrypted packet headers.	Accuracy
Use the ensemble methods when detecting DoS attacks in Network Intrusion Detection Systems	Bagging, Adaboost and Voting as ensembling qualifier. NaiveBayes,Random tree and KNN as classifiers	Optimized time complexity and has a metric of 98.76%	NaiveBayes and Random tree classifiers are not best Classifiers	F-score precision Accuracy
DDoS Attack Detection Method Based on Machine Learning.	Random forest algorithm Machine learning.	By using machine learning method for the DDoS attack has a good detection rate for the current popular DDoS attacks . It can extract the DDoS attacks out .	Takes time to detect the traffic characteristics with a large proportion.	Packets (TCP, UDP and ICMP) of the DDOS attack tools.
DDoS attack detection method based on improved KNN with DDoS	K-Nearest Neighbors (KNN) algorithm	The algorithms are identified the DDoS attack better, and the algorithms have achieved higher detection rates compared with	Does not work well with large dataset Does not work well with high dimensions Sensitive to noisy data, missing values and outliers	Accuracy

		the existing methods and solutions .		
DoS/DDoS Attack Detection Using Artificial Neural Networks	Artificail Neural Network	Detected known ddos attacks with full cent and with unknown it detected upto 95%	it cannot handle DDoS attacks that use encrypted packet headers.	Accuracy
A Survey on Slow DDoS Attack Detection Techniques	Six classifiers of random forest, KNN, logistic regression, SVM, and deep neural networks, CUSUM algorithm	The high detection rate of 99.905% in random forest. Average attack detection time of 32 seconds. Ease of implementation , Ease of detection because attacks stress the server's resources	Only slowloris attack was considered, Only slow header and slow POST were examined	Accuracy and precision
ML method for detecting DDoS attacks in the application layer	Neural Networks Genetic algorithm, Support vector machine (SVM), Fuzzy estimator	shows 95% of attacks generated by the tools have been detected effectively.	High time complexity	Accuracy
Server test for detecting Ddos Attcak: A Hybrid	SVM,NaiveBayes,KNN and Random frost	Less time complexity	Less accuracy of 93%	Accuracy

machine learning approach learning approach				
Smart Detection: An Approach for DoS/DDoS Attack Detection Using Machine Learning	Random Forest, Logistic Regression, AdaBoost, Stochastic gradient descent, and a perceptron.	i) Low false alarm rate ii) High accuracy of 96%	i)Need improved hit rate among attack classes ii)Automatic parameter calibration is missing	Accuraacy

PROPOSED ARCHITECTURE



DATASET USED

Application layer Dos Attack dataset from Kaggle which cosist of nearly 700000 rows and 78 columns with one class label.Class lable has three classifications Benign, DosHulk, Dosslowloris.

IMPLEMENTATION

Importing required libraries

```
from mpl_toolkits.mplot3d import Axes3D #For Basic ploting
from sklearn.preprocessing import StandardScaler #Preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn import preprocessing
# Preprocessing
from sklearn.naive_bayes import GaussianNB #import gaussian naivebayes model
from sklearn.tree import DecisionTreeClassifier #import Decision treeclassifier
from sklearn import metrics #Import scikit-learn metrics module foraccuracy calculation
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")
```

Writing a function for data visulization

```
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns 
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

preparing the data

```
nRowsRead = 1000 # specify No. of row. 'None' for whole data
# test_mosaic.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows
df1 = pd.read_csv('train_mosaic.csv', delimiter=',', nrows = nRowsRead)
df1.dataframeName = 'train_mosaic.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 1000 rows and 78 columns
```

```
df1.head()
```

	Destination_Port	Flow_Duration	Total_Fwd_Packets	Total_Backward_Packets	Total_Length_of_Fwd_Packets	Total_Length_of_Bwd_Packets	Fwd_Packet_Le
0	80	101168794	20	1	969	0	
1	60711	58	1	1	0	0	
2	53	31146	4	2	148	244	
3	80	254704	3	4	429	389	
4	443	11932077	12	16	5030	15703	

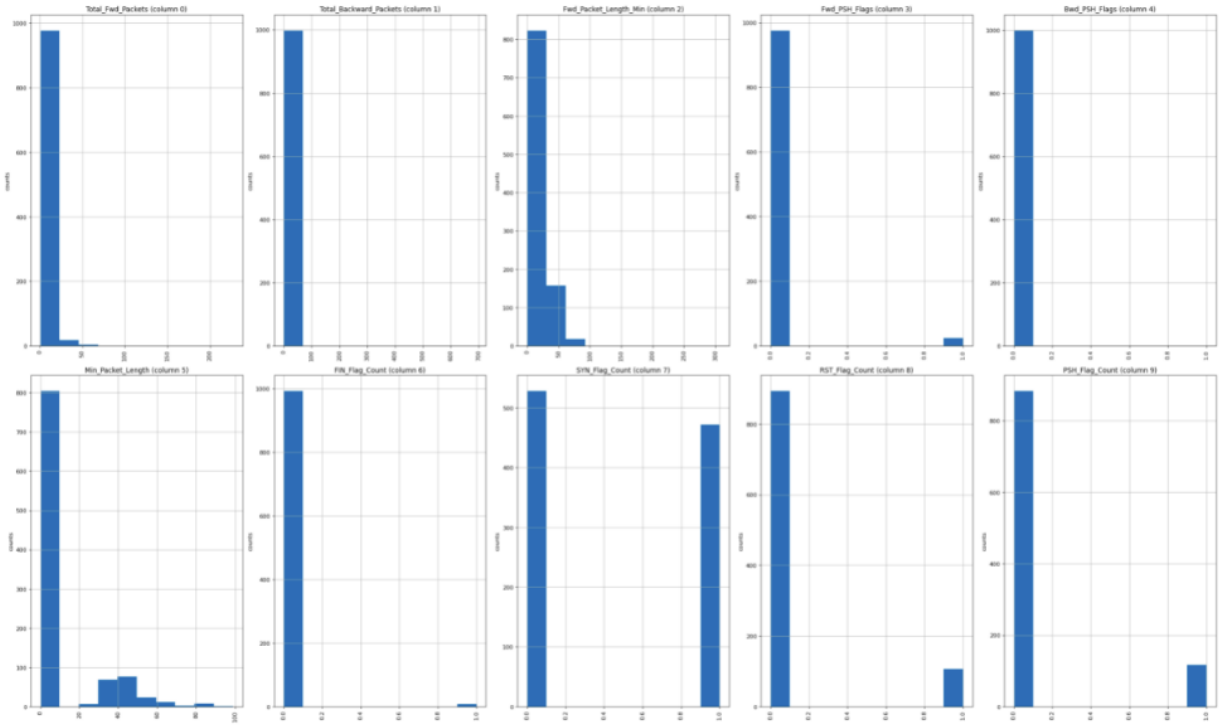
5 rows × 78 columns

```
df1.dtypes
```

```
Destination_Port                int64
Flow_Duration                   int64
Total_Fwd_Packets               int64
Total_Backward_Packets          int64
Total_Length_of_Fwd_Packets     int64
                                ...
Idle_Mean                     float64
Idle_Std                      float64
Idle_Max                      float64
Idle_Min                      float64
Label                          object
Length: 78, dtype: object
```

```
plotPerColumnDistribution(df1,10,5)
```

```
/tmp/ipykernel_179364/1562888048.py:9: MatplotlibDeprecationWarning: Passing non-integers as three-element positio
n specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(nGraphRow, nGraphPerRow, i + 1)
```




```
nRowsRead = None # specify No. of rows. 'None' for whole file
# train_mosaic.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows
df2 = pd.read_csv('test_mosaic.csv', delimiter=',', nrows = nRowsRead)
df2.dataframeName = 'test_mosaic.csv'
```

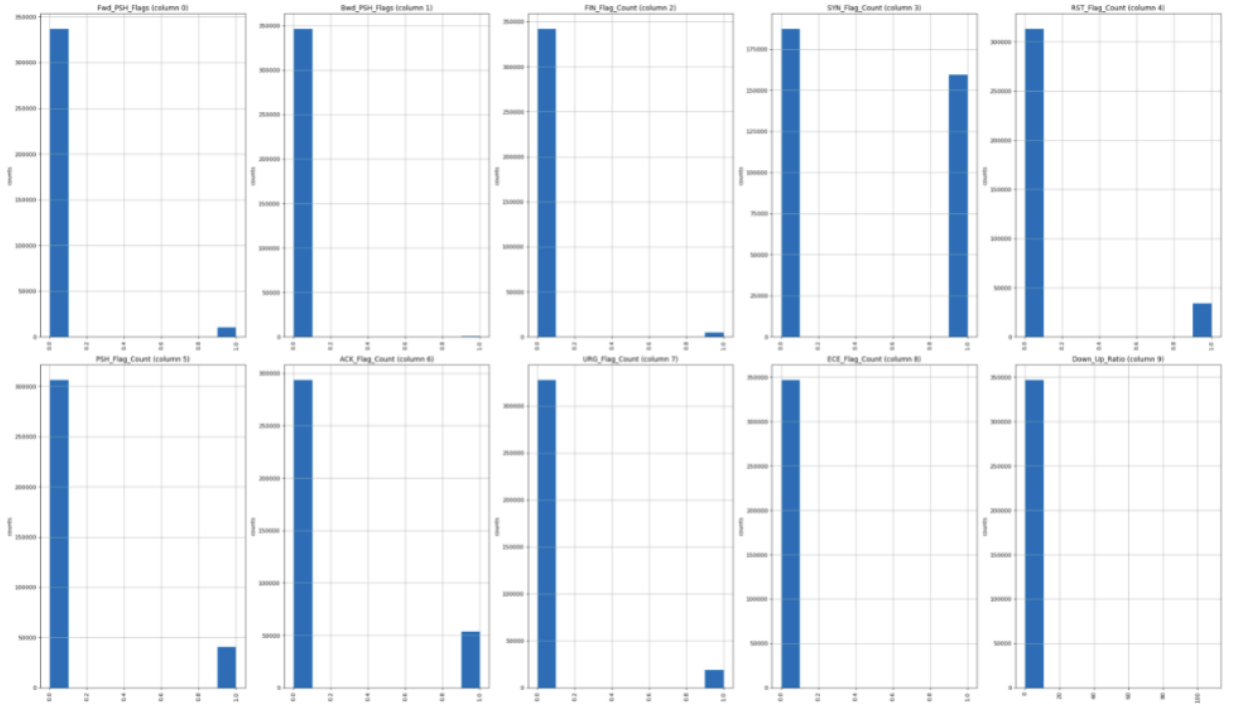
```
df2.head()
```

	Destination_Port	Flow_Duration	Total_Fwd_Packets	Total_Backward_Packets	Total_Length_of_Fwd_Packets	Total_Length_of_Bwd_Packets	Fwd_Packet_Le
0	53	87750	2	2	72	264	
1	53	31073	4	4	120	232	
2	80	41125329	8	1	387	0	
3	53	40633	4	4	140	508	
4	80	41920705	7	1	211	0	

5 rows × 78 columns

```
plotPerColumnDistribution(df2,10,5)
```

```
/tmp/ipykernel_179364/1562888048.py:9: MatplotlibDeprecationWarning: Passing non-integers as three-element positio
n specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(nGraphRow, nGraphPerRow, i + 1)
```




```
df = pd.concat([df1,df2])
df.shape
```

```
(347869, 78)
```

```
df.head()
```

	Destination_Port	Flow_Duration	Total_Fwd_Packets	Total_Backward_Packets	Total_Length_of_Fwd_Packets	Total_Length_of_Bwd_Packets	Fwd_Packet_Le
0	80	101168794	20	1	969	0	
1	60711	58	1	1	0	0	
2	53	31146	4	2	148	244	
3	80	254704	3	4	429	389	
4	443	11932077	12	16	5030	15703	

5 rows × 78 columns

```
X = df.drop('Label', axis=1)
y = df['Label']
```

```
scaler=MinMaxScaler((-1,1))
```

```
from imblearn.under_sampling import RandomUnderSampler
rus=RandomUnderSampler()
x_rus, y_rus = rus.fit_resample(X, y)
x=scaler.fit_transform(x_rus)
y = y_rus
```

```
x
```

```
array([[-0.99838252, -0.99999635, -0.99999047, ..., -1.        ,
        -1.        , -1.        ],
       [-0.99838252, -0.99999763, -0.99999047, ..., -1.        ,
        -1.        , -1.        ],
       [-0.99838252, -0.99947293, -0.99997142, ..., -1.        ,
        -1.        , -1.        ],
       ...,
       [-0.99755852,  0.09033917, -0.99991425, ..., -0.87199569,
        -0.58832318, -0.74925488],
       [-0.99755852,  0.05545158, -0.99991425, ..., -0.95801521,
        -0.6328046 , -0.6826287 ],
       [-0.99755852, -0.52094616, -0.99996189, ..., -1.        ,
        -0.52436658, -0.52436658]])
```

Splitting the data and training

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

Ensembling Technique:

```
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

```
print("Accuracy",metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy 0.999849402126442
```

```
clf = svm.SVC(kernel='linear') # Linear Kernel
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9968976838047047
```

```
clf=RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
```

```
rf1 = RandomForestClassifier(n_estimators=100)
dt1 = DecisionTreeClassifier()
svm1 = svm.SVC(kernel = 'linear')
```

```
from sklearn. ensemble import VotingClassifier
evc = VotingClassifier( estimators= [('rf1',rf1),('dt1',dt1),('svm1',svm1)],verbose=True)
```

```
evc.fit(pd.DataFrame(X_train).iloc[1:],pd.DataFrame(y_train).iloc[1:])
```

```
[Voting] ..................... (1 of 3) Processing rf1, total=  17.4s
[Voting] ..................... (2 of 3) Processing dt1, total=   2.3s
[Voting] ..................... (3 of 3) Processing svm1, total= 1.4min

VotingClassifier(estimators=[('rf1', RandomForestClassifier()),
                             ('dt1', DecisionTreeClassifier()),
                             ('svm1', SVC(kernel='linear'))],
                 verbose=True)
```

```
print("Accuracy is {}".format(evc.score(X_test, y_test)))
```

```
Accuracy is 0.999849402126442
```

The accuracy of this process is 99.999963%, most accurate method so far.

Since DDoS attacks can bog down your entire IT infrastructure, your employees are not able to access internal resources such as email, VoIP, and other crucial resources for business processes.

From the analysis we can conclude that many IT companies that use DDOS attack tools to test server stress prefer xerxes tool which is most powerful tool compared to slowloris and hulk for the attack and to improve the security.