Information Security Management (CSE3502)

Review 2

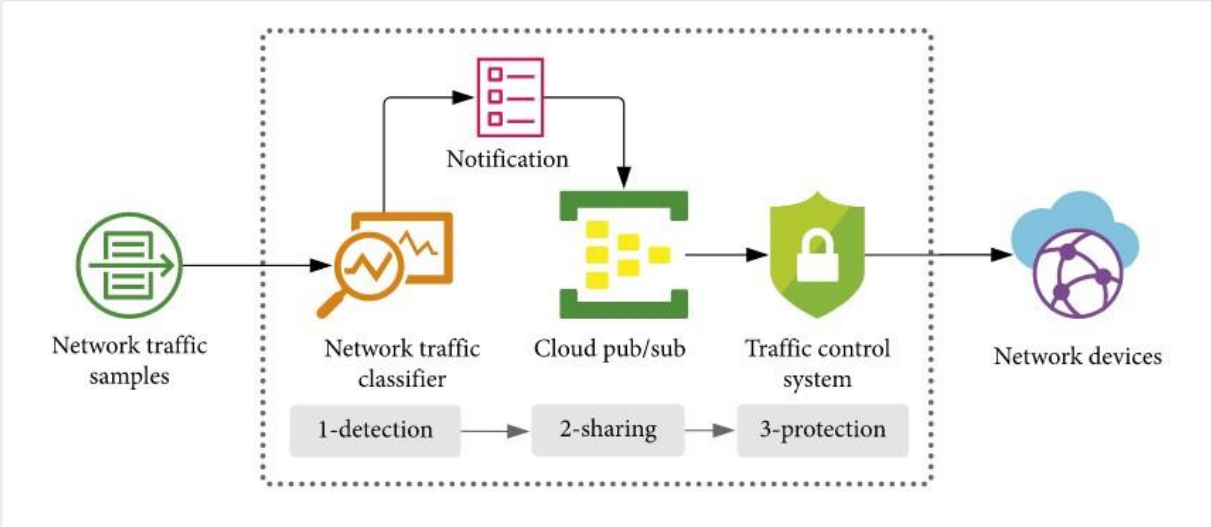
TITLE: DETECTING DDOS ATTACK USING MACHINE LEARNING

By- Gauransh Arora (18BIT0393) Ayushi Gupta (18BIT0367)

1. Design/Architecture of the Proposed System/Model

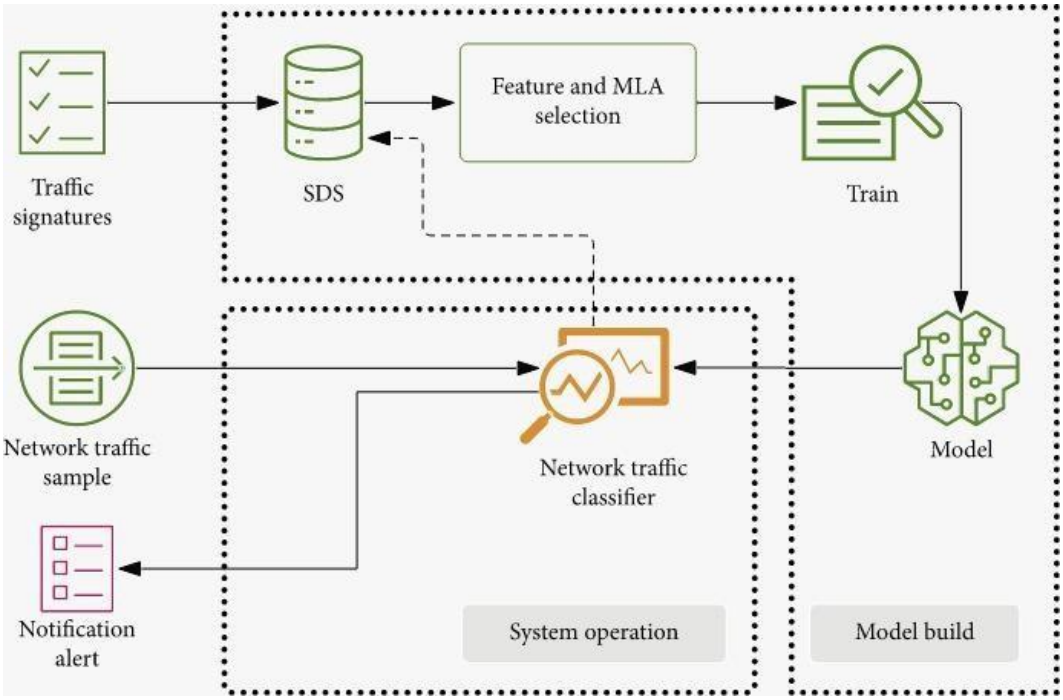
1.1 Low Level Diagram

Smart Detection is designed to combat DDoS attacks on the Internet in a modern collaborative way. In this approach, the system collects network traffic samples and classifies them. Attack notification messages are shared using a cloud platform for convenient use by traffic control protection systems.

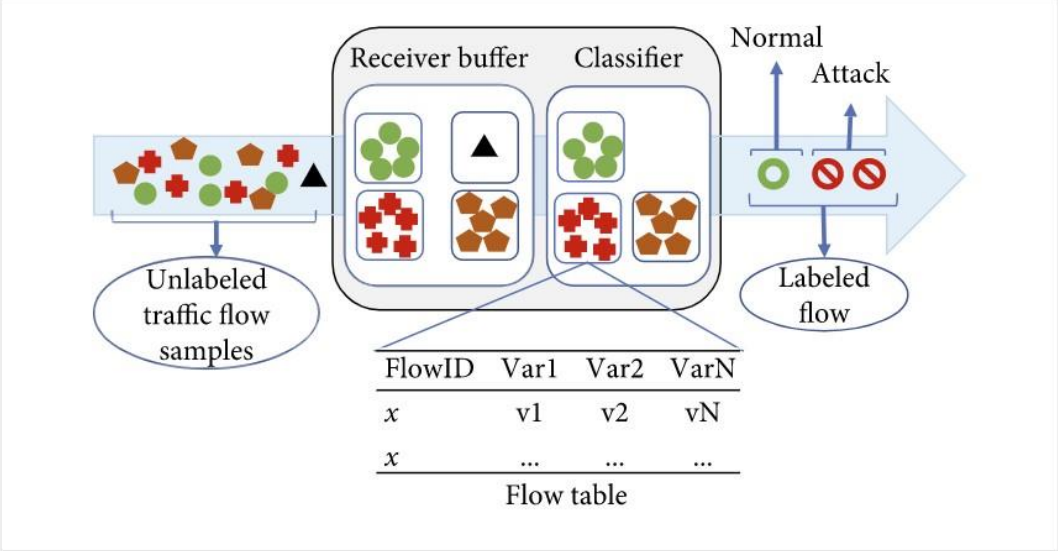


1.2 High Level Diagram

The core of the detection system consists of a Signature Dataset (SDS) (dataset being used form http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html) and a machine learning algorithm (MLA).



First, normal traffic and DDoS signatures were extracted, labeled, and stored in a database. SDS was then created using feature selection techniques. Finally, the most accurate MLA was selected, trained, and loaded into the traffic classification system.

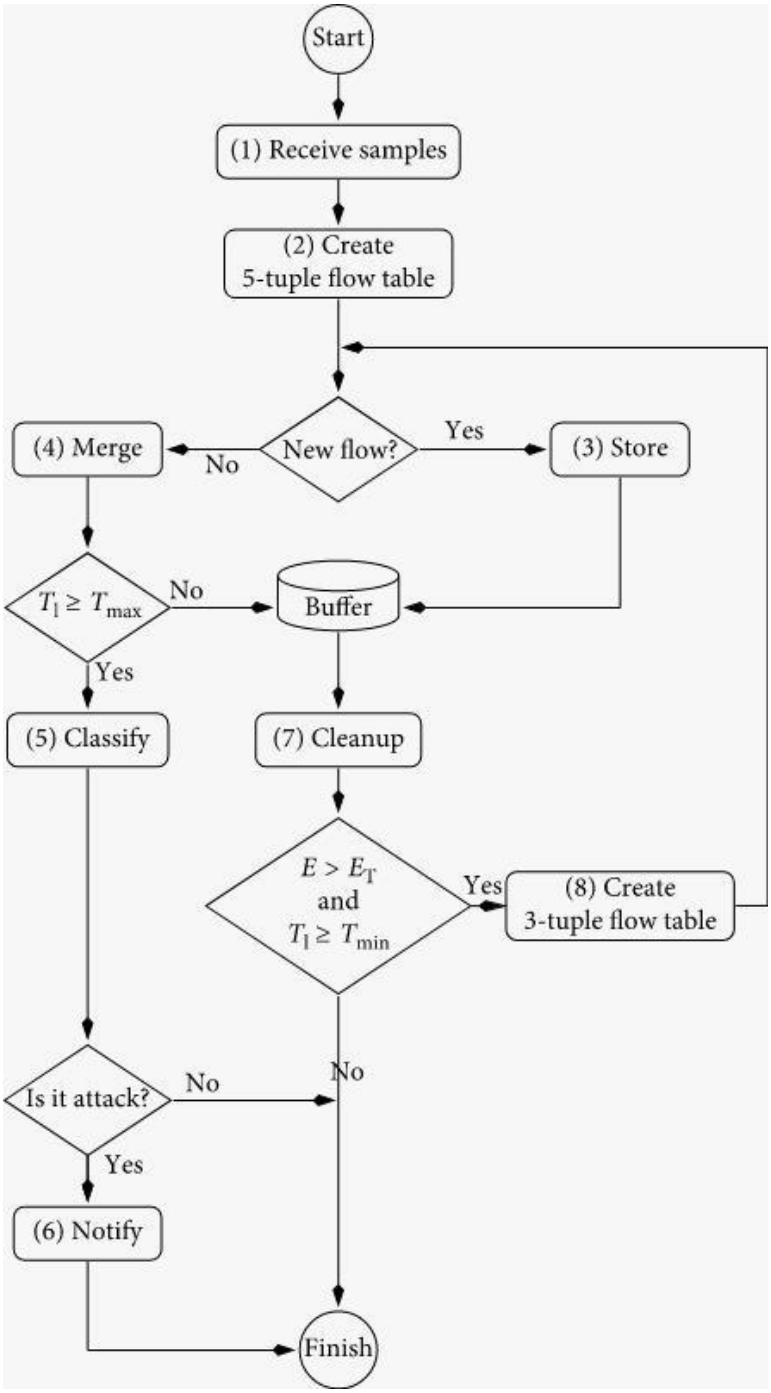


The architecture of the detection system was designed to work with samples of network traffic provided by industrial standard traffic sampling protocols,

collected from network devices. The unlabeled samples are received and grouped in flow tables in the receiver buffer. Thus, when the table length is greater than or equal to the reference value, they are presented to the classifier responsible for labeling them. If the flow table expires, it may be processed one more time. The occurrence of small flow tables is higher at lower sampling rates or under some types of DoS attacks, e.g., SYN flood attacks.

1.3 Data Flow Diagram

The complete algorithm of the detection system is summarized in Figure 4. During each cycle of the detection process, traffic samples are received and stored in a flow table. For each new flow, a unique identifier (FlowID) is calculated based on the 5-tuple (src_IP, dst_IP, src_port, dst_port, and transport_protocol) in steps 1 and 2. If this is a new flow, i.e., there is not any other flow table stored with the same FlowID, the flow table is registered in a shared memory buffer. Otherwise, if there is a flow table registered with the same FlowID such as the previously calculated one, the data of the new flow will be merged with the data in the existing flow table in steps 3 and 4. After the merging operation, if the table length is greater than or equal to the reference value (), the flow table is classified, and if it is found to be an attack, a notification is emitted. Otherwise, it is inserted back into the shared memory buffer. Meanwhile, in step 7, the cleanup task looks for expired flow tables in the shared buffer, i.e., flow tables that exceed the expiration time of the system (). For each expired flow table, the system checks the table length. If the flow table length is less than or equal to the minimum reference value (), this flow table will be processed by step 8. A new FlowID is calculated using the 3-tuple (src_IP, dst_IP, and transport_protocol), as the flow table is routed back to steps 3 and 4.


Start
(1) Receive samples
(2) Create 5-tuple flow table
New flow?
Yes
No
(3) Store
(4) Merge
$T_l \geq T_{max}$
No
Buffer
Yes
(5) Classify
(7) Cleanup
$E > E_T$ and $T_l \geq T_{min}$
Yes
(8) Create 3-tuple flow table
Is it attack?
No
No
Yes
(6) Notify
Finish

2. Modules' description and Implementation

Modules:

Icmp_attack.ipynb: This is the script which implements algorithm for detection of Ddos attack in ICMP packet which is written by Gauransh.

Tcp_syn_attack.ipynb: This is the script which implements algorithm for detection of Ddos attack in TCP packet which is written by Ayushi.

Udp_attack.ipynb: This is the script which implements algorithm for detection of Ddos attack in UDP packet which is written by Gauransh.

WORK DISTRIBUTION :
Now Ayushi is working on pickle library , how can we store trained model.And Gauransh will combine all the code in just one to form the final code.

Future Modules:

i. Train.py: This will train the models that are created to find ddos attack through tcp , icmp , and udp packages. It will require around 70 percent of the dataset. And will store the instant trained model for further use where time required will be the least.

ii. Test.py: used to extract the pretrained model and test the model on the parameters given with the 30 percent of the dataset.

Models used:

i. Logistic regression: Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).
It is used to obtain odds ratio in the presence of more than one explanatory variable. The procedure is quite similar to multiple linear regression, with the exception that the response variable is binomial. The result is the impact of each variable on the odds ratio of the observed event of interest.

ii. KNeighboursClassifier: KNeighborsClassifier implements classification based on voting by nearest k-neighbors of target point, t, while RadiusNeighborsClassifier implements classification based on all neighborhood points within a fixed radius, r, of target point, t. It works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

iii. MLPClassifier: MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.
It is a class of feedforward artificial neural network (ANN). MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

iv. Decision Tree Classifier: Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too.
The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data (training data). The understanding level of Decision Trees algorithm is so easy compared with other classification algorithms.

Implementation:

Icmp_attack.ipynb

In [2]:
```
#ICMP Flood attack models
#Author: GAURANSH ARORA
```

In [3]:
```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
```

In [4]:
```
colnames = ["duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fragment","urgent","hot","num_faile
```

In [5]:
```
len(colnames)
```

Out[5]:
```
42
```

In [6]:
```
df = pd.read_csv("./corrected.csv",header=None,names=colnames)
```

In [7]:
```
df.head(10)
```

Out[7]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host
0	0	udp	private	SF	105	146	0	0	0	0	...	254	1.00	
1	0	udp	private	SF	105	146	0	0	0	0	...	254	1.00	
2	0	udp	private	SF	105	146	0	0	0	0	...	254	1.00	

In [8]:
```
df.shape
```

Out[8]:
```
(311029, 42)
```

In [9]:
```
#below csv file is a revised one sent to the same folder
df.to_csv("revised_kddcup_dataset.csv")
```

n [10]:
```
#extracting the icmp packets from our dataset
icmp_df = df[df.loc[:,"protocol_type"] == "icmp"]
```

n [11]:
```
icmp_df.isnull().sum()
#none of the values in dataset are null
```

ut[11]:
```
duration                0
protocol_type           0
service                 0
flag                    0
src_bytes               0
dst_bytes               0
land                    0
wrong_fragment          0
urgent                  0
hot                     0
num_failed_logins       0
logged_in               0
num_compromised         0
root_shell              0
```

In [12]:
```
#sending icmp attack file as csv dataset
icmp_df.to_csv("revised_icmp_dataset.csv")
```

In [13]:
```
icmp_df.head()
```

Out[13]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_hos
82	0	icmp	eco_i	SF	30	0	0	0	0	0	...	3	0.01	
156	0	icmp	ecr_i	SF	30	0	0	0	0	0	...	75	1.00	
406	0	icmp	ecr_i	SF	30	0	0	0	0	0	...	98	1.00	
629	0	icmp	ecr_i	SF	30	0	0	0	0	0	...	120	1.00	
767	0	icmp	eco_i	SF	30	0	0	0	0	0	...	2	0.01	

5 rows × 42 columns

In [14]:
```
#I will be extracting all the important features as a "priori" for preprocessing
features = ["duration","service","src_bytes","wrong_fragment","count","urgent","num_compromised","srv_count"]
target = "result"
```

In [15]:
```
X = icmp_df.loc[:,features]
y = icmp_df.loc[:,target]
```

In [17]:
```
classes = np.unique(y)
print(classes)
```

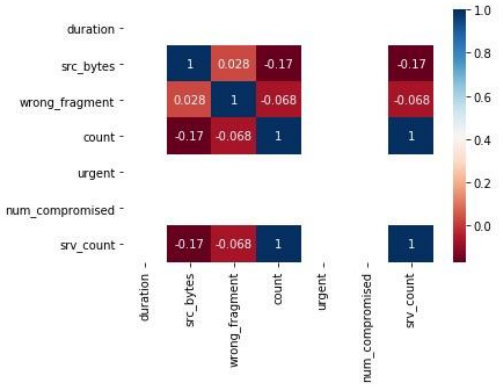
```
['ipsweep.' 'multihop.' 'normal.' 'pod.' 'saint.' 'satan.' 'smurf.'
 'snmpguess.']
```

In [18]:
```
#replacing all classes of attack with 1 and normal result with 0 in our icmp_df
for i in range(len(classes)):
    if i == 2:
        icmp_df = icmp_df.replace(classes[i], 0)
    else:
        icmp_df = icmp_df.replace(classes[i], 1)

#turning the service attribute to categorical values
icmp_df=icmp_df.replace("eco_i",-0.1)
icmp_df=icmp_df.replace("ecr_i",0.0)
icmp_df=icmp_df.replace("tim_i",0.1)
icmp_df=icmp_df.replace("urp_i",0.2)
```

In [19]:
```
y = icmp_df.loc[:,target]
```

In [20]:
```
#I selected certain features but I will have to find some covariance between them so I will plot a covariance heatmap
sns.heatmap(X.corr(), annot=True,cmap="RdBu")
plt.plot()
#the data as seen is highly uncorrelated as most of it is one valued such as the duration one.
```

Out[20]: []



In [29]:
```
X = icmp_df.loc[:,features]
y = icmp_df.loc[:,target]
X.head()
```

In [29]:
```
X = icmp_df.loc[:,features]
y = icmp_df.loc[:,target]
X.head()
```

Out[29]:

	duration	service	src_bytes	wrong_fragment	count	urgent	num_compromised	srv_count
82	0	-0.1	30	0	2	0	0	2
156	0	0.0	30	0	2	0	0	2
406	0	0.0	30	0	2	0	0	2
629	0	0.0	30	0	1	0	0	1
767	0	-0.1	30	0	3	0	0	1

In [30]:
```
print(list(X.loc[629,:])) #7 input features
```

```
[0.0, 0.0, 30.0, 0.0, 1.0, 0.0, 0.0, 1.0]
```

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)
```

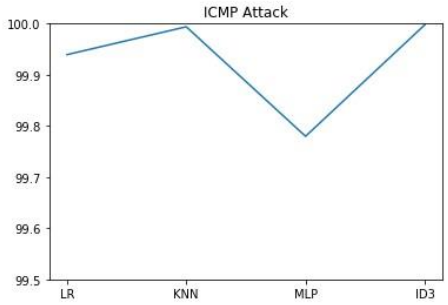
In [24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier

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

In [25]:
```
models = [LogisticRegression(), KNeighborsClassifier(n_neighbors=3),MLPClassifier(alpha=0.005),DecisionTreeClassifier()]
classifiers = ["LR", "KNN","MLP","ID3"]
scores = []
```

In [26]:
```
for model in models:
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)*100
    scores.append(score)
    print("Accuracy of the model is: ", score)
    conf_matrix = confusion_matrix(y_test,y_pred)
    report = classification_report(y_test,y_pred)
    print("Confusion Matrix:\n",conf_matrix)
    print("Report:\n",report)
    print("\n==============***===============")
```

```
Accuracy of the model is:  99.93938291810632
Confusion Matrix:
 [[   81    25]
 [    5 49380]]
Report:
              precision    recall  f1-score   support

           0       0.94      0.76      0.84       106
           1       1.00      1.00      1.00     49385

   micro avg       1.00      1.00      1.00     49491
   macro avg       0.97      0.88      0.92     49491
weighted avg       1.00      1.00      1.00     49491


==============***===============
Accuracy of the model is:  99.99393829181064
Confusion Matrix:
 [[  104     2]
 [    1 49384]]
Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       106
           1       1.00      1.00      1.00     49385

   micro avg       1.00      1.00      1.00     49491
   macro avg       1.00      0.99      0.99     49491
weighted avg       1.00      1.00      1.00     49491
```

```
In [28]: plt.plot(classifiers,scores)
         plt.title("ICMP Attack")
         plt.ylim(99.5,100)
         plt.show()
```



Similarly script is written for Tcp_syn_attack.ipynb, And Udp _attack.ipynb

Our single code for all packages can be seen here:
https://colab.research.google.com/drive/16vIZkPfsnFc9Bvvqx_5Gm8tPlDs-m6A5?usp=sharing

THANK YOU