Detection of Brute-Force Attacks in End-to-End Encrypted Network Traffic

Team Members:

Manisha Mahapatra [CS22MTECH14009]
Julakuntla Madhuri[CS20BTECH11023]

Problem Statement:

Intrusion detection systems (IDSs) are extensively used by organisations to safeguard their IT systems against assaults, but they are constrained by the rising usage of encrypted communication on the internet, which makes network IDSs (NIDSs) less effective.

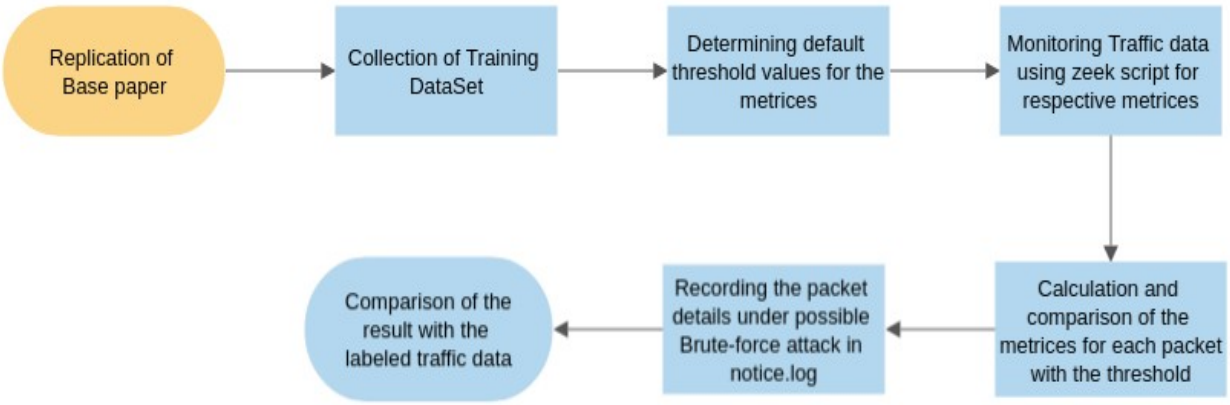
To address this constraint, organisations can utilise man-in-the-middle proxies in conjunction with NIDSs to preserve some network traffic analysis capabilities for encrypted data, but this involves issuing TLS certificates that clients can trust. However, network infrastructure providers have limited influence over destination systems, and VPNs and Tor exit nodes may be incentivized to avoid assaults in encrypted data.

This paper provides a novel method for detecting brute-force assaults in encrypted network traffic. The method employs five novel indications to quantify traffic patterns vital for detecting brute-force attacks without the need for decryption of encrypted payload.

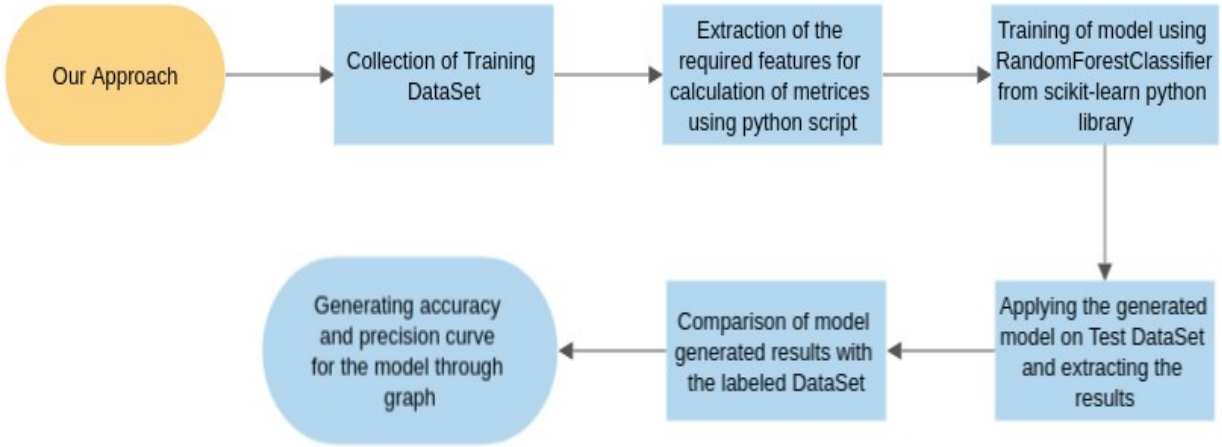
Project Description:

We aim to showcase mechanism for Brute-force attack detection on the basis of specified metrices and their threshold obtained through analysis of known traffic dataset, such that decryption of the encrypted data can be avoided for the purpose of detection.
The setup has been implemented through two approaches:

A) Implementing the setup using zeek scripting language which directly monitors the traffic from the pcap file and detects on the basis of pre-defined thresholds.



B) Implementing using python by training a classifier model using ML algorithm for detecting possible Brute-force attack in the captured network traffic.



Network based Intrusion Detection System (NIDS)

A Network-based Intrusion Detection System (**NIDS**) is a type of security system that analyses network traffic for potential security threats and identifies and responds to them. **NIDS** is designed to analyse traffic patterns and detect harmful activities such as unauthorised access attempts, malware infections, and other sorts of assaults.

NIDS are commonly used as standalone devices or as part of a wider security system. It works by inspecting all network traffic passing across the network segment being watched. This includes all incoming and outgoing network traffic from all devices.

When an intrusion detection system (**NIDS**) detects a potential security threat, it can take a number of measures, depending on how it is configured. It may, for example, notify security staff, block the offending traffic, or take other automatic actions.

Overall, network intrusion detection systems (**NIDS**) are an important component of any complete network security strategy because they provide an additional layer of defence against a wide range of cyber attacks.

When it comes to identifying and analysing encrypted communication, **NIDS** deal with a number of setbacks. Encrypted data is intended to protect sensitive information from being intercepted and read by unauthorised users, but it also makes monitoring and analysing traffic more difficult.

One of the most difficult aspects of identifying encrypted traffic is that the content of the transmission is invisible to **NIDS**. The contents of network packets are normally analysed by **NIDS**, although encrypted traffic is designed to prevent anyone from reading the contents of those packets. As a result, NIDS cannot identify or analyse encrypted communication in the same way that it can detect and analyse unencrypted traffic.

Project Implementation Details

Metrices for Brute-force attack detection :

- $\mathbf{m_{esr}(t)}$ = no. of equally sized response packets within a specific time interval t, assuming that response from server to authentication failure is always of same length.
- $\mathbf{m_{ssr}(t, d)}$ = no. of similarly sized responses within a specific time interval t, assuming that the size of responses differs at most by d bytes, i.e., for services where there are minor differences in responses.
- $\mathbf{m_{cc}(t)}$ = no. of TCP connections within a specific time interval t (connection count), assuming that separate connection used for each authentication attempt.
- $\mathbf{m_{pc}(t)}$ = no. of TCP packets within a specific time interval t (packet count), assuming that multiple authentication attempts are performed via the same TCP connection.
- $\mathbf{m_{sc}(t, s)}$ = no. of short-living TCP connections within a specific time interval t (short connection), i. e., connections with a duration of at most s seconds.

Modes of applying metrices :

The attack detection method based on these metrics can be applied in three different modes:

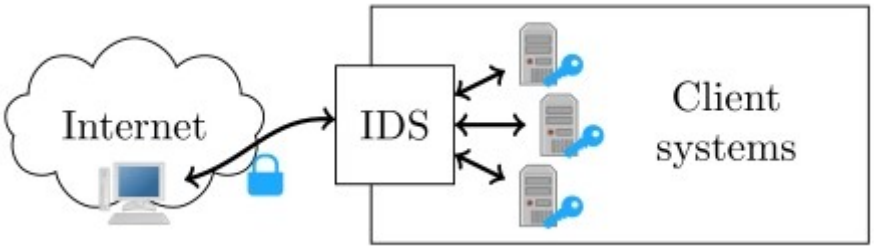
- **Source-oriented** (applied per source) : classifies traffic sources as attackers if the metric's value for that source exceeds a threshold.
- **Destination-oriented** (applied per destination) : classifies destination as being attacked, especially useful when information on traffic sources is not available, for e.g., in detecting distributed attacks.
- **Both source- and destination-oriented** : allows to detect distributed brute-force attacks as well as to track the origins of attacks performed from individual systems.

Collection of Traffic Data :

- Requirement of the traffic dataset pcap and csv was to include Brute-force attacks on the packets sents over different protocols including HTTP and FTP (considered in this project).
- For this purpose, we have considered **CICIDS2017** dataset which contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes CSV files for machine and deep learning purpose.
- Thursday, July 6, 2017 (Morning) -> **Web Attack – Brute Force** (9:20 – 10 a.m.)
- Attacker: **Kali, 205.174.165.73**
- Victim: **WebServer Ubuntu, 205.174.165.68** (Local IP192.168.10.50)
- Initial evaluation of metrics on synthetic traffic with existing dataset.

Implementation of base paper method

We have implemented mechanism for detection of Brute-force attack as per the approach described in the base paper using Zeek scripting language.
This involved hardcoding of threshold values for the metrices on the basis of which the traffic packets were compared to detect Brute-force attack.
As per the setup, traffic etween clients is encrypted. The Intrusion Detection System analyzes it without decrypting the traffic.
Helps to detect attacks and avoid higher computational cost & overhead from decypting the Brute-force traffic.



> Approach_1 > Destination_oriented > Connection_Count > zk1.zeek

```
#!/usr/bin/env zeek
#dest-oriented_connection_count_https.zeek

module HTTPSConnectionCountBruteForce;

export {
  # Protocol name
  const proto = "HTTPS";

  # Service port
  const service_port = 443/tcp;
}

@load base/frameworks/notice
@load base/frameworks/sumstats

export {
  redef enum Notice::Type += {
    ## Indicates that a host has been identified as performing a brute force attack
    Brute_Forcing,
  };

  # Max-length packet
  const full_packet_len = 16406;
}

export {
  # Duration of a single epoch to count
  const epoch_len = 10min;

  # The threshold for the number of connections with the same origin/destination
  const brute_threshold = 10;
}

event zeek_init() &priority=-101 {
  # send a notice to make sure notice.log is created
  print(fmt("connection-count brute-force script loaded for proto %s (%s)", proto, service_port));
  NOTICE([$note=Brute_Forcing,
          $msg=fmt("connection-count brute-force script loaded for proto %s (%s)", proto, service_port)]);
}
```

```
event zeek_init() &priority=5 {
  local reducer = SumStats::Reducer($stream="HTTPSConnectionCountBruteForce connection observed", $apply=set(SumStats::SUM));

  SumStats::create([
    $name = "counting HTTPSConnectionCountBruteForce connections", $epoch = epoch_len, $reducers = set(reducer),
    $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) = {
      local sum = result["HTTPSConnectionCountBruteForce connection observed"]$sum;
      if (sum >= brute_threshold) {
        NOTICE([$note=Brute_Forcing,
                $msg=fmt("%s appears to be brute-forcing/brute-forced %s (seen %f connections).", key$host, proto, sum),
                $src=key$host]);
      }
    }]);
}

event connection_state_remove(c: connection) {
  # ignore non-service packets
  if (c$id$resp_p != service_port) return;

  SumStats::observe("HTTPSConnectionCountBruteForce connection observed",
    SumStats::Key($host=c$id$resp_h),
    SumStats::Observation($num=1));
}
```

- The script defines the **HTTPSConnectionCountBruteForce** module, which is in charge of detecting potential brute force attacks against HTTPS connections.
- Several constants and global variables exported by the module are used throughout the script, including the protocol name (**HTTPS**), service port (**443/tcp**), and observation epoch period (10 minutes).
- The module also specifies the **Brute_Forcing** notification type, which is used to signal that a host has been recognised as conducting a brute force attack.
- The script contains two initialization events that are executed when Zeek is launched. The first event sends a notice to ensure that the notice log is produced, and the second event builds a **SumStats** reducer to count the number of HTTPS connections seen during each epoch.
- The second initialization event's **SumStats** reducer counts the number of HTTPS connections with the same origin/destination seen over each epoch. If the number of connections exceeds a predefined threshold (brute_threshold = 10), the script generates a security notification indicating the possibility of a brute force attack.
- A **connection_state_remove** event is also included in the script, which is fired once a connection is deleted. This event determines if the connection is HTTPS and updates the **SumStats** counter for that connection. If the number of connections with the same origin/destination exceeds the defined threshold, the script issues a security warning signalling a possible brute force attack.
- The script creates a **SumStats** stream using the built-in Zeek **SumStats** framework to tally the number of HTTPS connections seen across each epoch. The **SumStats** framework is built to handle big amounts of data efficiently and is ideal for monitoring network traffic over lengthy periods of time.
- The script uses the built-in Zeek Notice framework to generate security notices. Notices can be customised with unique phrases, severity levels, and other information to alert security administrators to potential security concerns.

Similar Implementation has been done in zeek for the remaining 4 metrices and seperate implementation for the 2 modes. This has been done by changing below shown parameter i.e. resp_h (destination-oriented) and orig_h (origin).

```
event connection_state_remove(c: connection) {
  # ignore non-service packets
  if (c$id$resp_p != service_port) return;

  SumStats::observe("HTTPSConnectionCountBruteForce connection observed",
    SumStats::Key($host=c$id$resp_h),
    SumStats::Observation($num=1));
}
```

```
event connection_state_remove(c: connection) {
  # ignore non-service packets
  if (c$id$resp_p != service_port) return;

  # orig_h for origin oriented
  SumStats::observe("HTTPSConnectionCountBruteForce connection observed",
    SumStats::Key($host=c$id$orig_h),
    SumStats::Observation($num=1));
}
```

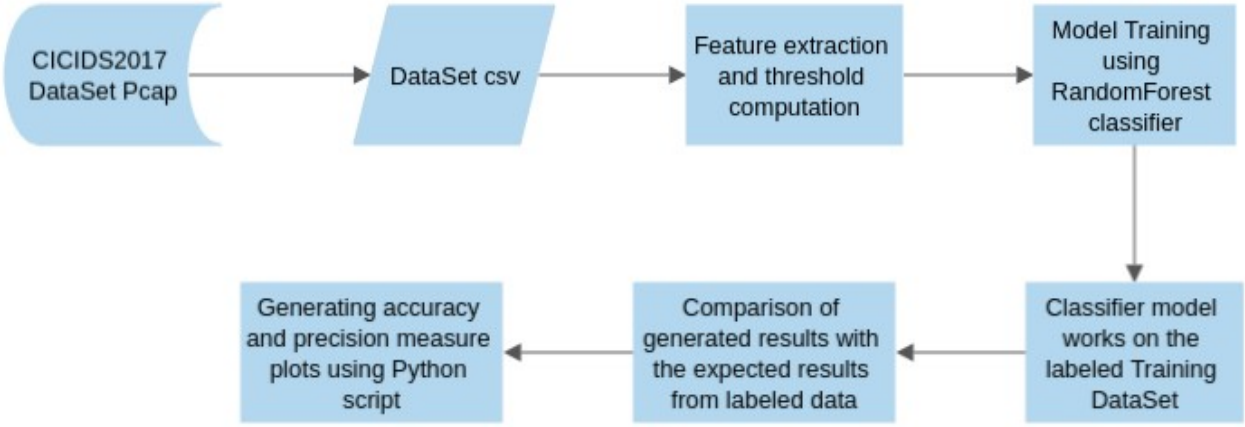
Challenges Faced

- Implemted zeek setup and related dependencies with python. To use Zeek with Python, installed Zeek Python API i.e. pyzeek. Setup not successful due to pip issue. The same was not avilable in conda.

```
  File "/usr/local/lib/python3.8/dist-packages/pyOpenSSL-23.1.1-py3.8.egg/OpenSS
L/__init__.py", line 8, in <module>
    from OpenSSL import SSL, crypto
  File "/usr/local/lib/python3.8/dist-packages/pyOpenSSL-23.1.1-py3.8.egg/OpenSS
L/SSL.py", line 19, in <module>
    from OpenSSL.crypto import (
  File "/usr/local/lib/python3.8/dist-packages/pyOpenSSL-23.1.1-py3.8.egg/OpenSS
L/crypto.py", line 3258, in <module>
    utils.deprecated(
TypeError: deprecated() got an unexpected keyword argument 'name'
```

- Used VM for setting up the environment (with Zeek and Python integration), but the configuration still faced issues due to no **prof.log , packet_filter.log, loaded_scripts.log**
  Updated **'/opt/zeek/etc/zeekctl.cfg'** to enable Logging and set below parameters to '1':
  CompressLogs, LogAscii, LogBinary
- Checking Zeek scripts to ensure that logging is enabled for the specific logs. The paths to the relevant script files for each log:
  - **prof.log: /opt/zeek/share/zeek/site/local.zeek**
  - **packet_filter.log: /opt/zeek/share/zeek/site/local.zeek**
  - **loaded_scripts.log: /opt/zeek/share/zeek/site/local.zeek**
- Setting up PATH variable at **~/.bashrc** using super-user, still the integration issue could not be resolved
- Similar issue observed for virtual environment in local system using conda:
  (myenv)
  manisha@manisha-HP-Laptop-15-bs1xx:~/Documents/Sem2/NS/TermProject$ zeek zk1.zeek
  fatal error in ./zk1.zeek, line 16: can't find python3
- Issue with **CICIDS2017** dataset PCAP due to large size (10Gb), could not be handled with WireShark. Using tcpdump, this was segmented into smaller PCAPs.
- Format issue with captured Traffic Data, resolved by conversion into csv format when using our modified Python script.

Implementation of modification based on our approach




```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
import joblib
import csv
import numpy as np

def training_Model():
    print("-----------tM1----------")
    # Load the network traffic data CSV file
    df = pd.read_csv('network_traffic_data.csv')

    # Select only the relevant columns for feature extraction
    relevant_columns = ['DestinationPort', 'TotalFwdPackets', 'TotalBackwardPackets', 'TotalLengthofFwdPackets', 'TotalLengthofBwdPackets', 'FlowDuration']

    # Filter only the rows with the label "Web Attack  Brute Force"
    df = df[df['Label'] == 'Web Attack  Brute Force']

    # Set the time interval t in seconds
    t = 10

    # Extract the relevant features
    df['Number of equally sized response packets'] = df['TotalBackwardPackets'] // t
    df['Number of similarly sized responses'] = df.groupby('TotalLengthofFwdPackets')['TotalBackwardPackets'].transform('count') // t
    df['Number of TCP connections'] = df.groupby('DestinationPort')['DestinationPort'].transform('count') // t
    df['Number of TCP packets'] = df.groupby('DestinationPort')['TotalFwdPackets'].transform('sum') // t
    df['Number of short-living TCP connections'] = df[df['FlowDuration'] < 2*t].groupby('DestinationPort')['DestinationPort'].transform('count')

    # Select the relevant features for training the classifier
    features = ['Number of equally sized response packets', 'Number of similarly sized responses', 'Number of TCP connections', 'Number of TCP packets']
    X = df[features]
    y = df['Label']

    # Split the data into training and testing sets, only if there are instances with the label "Web Attack - Brute Force"
    if df.shape[0] > 0:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Train a Random Forest classifier
        clf = RandomForestClassifier()
        clf.fit(X_train, y_train)

        # Save the trained model to a file
        joblib.dump(clf, 'bruteforce_attack_model.joblib')
    print("-----------tM2----------")
```

```python
def feature_Extraction():
    print("-----------fE1----------")
    # Load the network traffic data CSV file
    df = pd.read_csv('network_traffic_data.csv')
    # Select only the relevant columns for feature extraction
    relevant_columns = ['DestinationPort', 'TotalFwdPackets', 'TotalBackwardPackets', 'TotalLengthofFwdPackets', 'TotalLengthofBwdPackets', 'FlowDuration']
    # Filter only the rows with the label "Web Attack  Brute Force"
    df = df[df['Label'] == 'Web Attack  Brute Force']
    # Set the time interval t in seconds
    t = 10

    # Extract the relevant features
    df['Number of equally sized response packets'] = df['TotalBackwardPackets'] // t
    df['Number of similarly sized responses'] = df.groupby('TotalLengthofFwdPackets')['TotalBackwardPackets'].transform('count') // t
    df['Number of TCP connections'] = df.groupby('DestinationPort')['DestinationPort'].transform('count') // t
    df['Number of TCP packets'] = df.groupby('DestinationPort')['TotalFwdPackets'].transform('sum') // t
    df['Number of short-living TCP connections'] = df[df['FlowDuration'] < 2*t].groupby('DestinationPort')['DestinationPort'].transform('count')
    # Select the relevant features for training the classifier
    features = ['Number of equally sized response packets', 'Number of similarly sized responses', 'Number of TCP connections', 'Number of TCP packets', 'Numb
    # Save the relevant features to a new CSV file for training the classifier
    df[features].to_csv('bruteforce_attack_features.csv', index=False)
    print("-----------fE2----------")
```

This code defines three main functions: training_Model, feature_Extraction, and testing_Model.

training_Model function:

- Reads a CSV file containing network traffic data.
- Selects relevant columns from the data for feature extraction.
- Filters only rows labeled as "**Web Attack – Brute Force**".
- Sets a time interval t in seconds.
- Extracts relevant features from the data, using these features for training the Random Forest classifier.
- Splits the data into training and testing sets.
- Trains a Random Forest classifier using the training set.
- Saves the trained model to a file.

feature_Extraction function:

- Reads a CSV file containing network traffic data.
- Selects relevant columns from the data for feature extraction.
- Filters only rows labeled as "**Web Attack – Brute Force**".
- Sets a time interval t in seconds.
- Extracts relevant features from the data.
- Saves the relevant features to a new CSV file for training the classifier.

testing_Model function:

- Loads a trained model from a file.
- Reads a CSV file containing network traffic data.
- Filters only rows labeled as "**Web Attack – Brute Force**".
- Sets a time interval t in seconds.
- Extracts relevant features from the data.
- Uses the trained model to predict whether each instance of data is a "**Web Attack – Brute Force**".

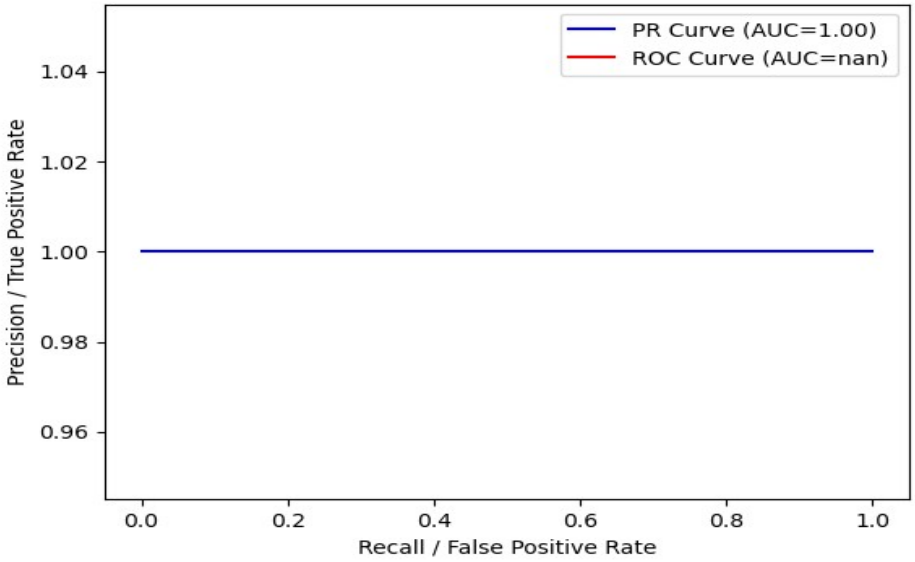
In addition there is also another function for conversion of input pcap file into csv file for further processing on it with the help of the script.

Performance Metrices

We have used below parameter for computing performance metrices of our implementation.

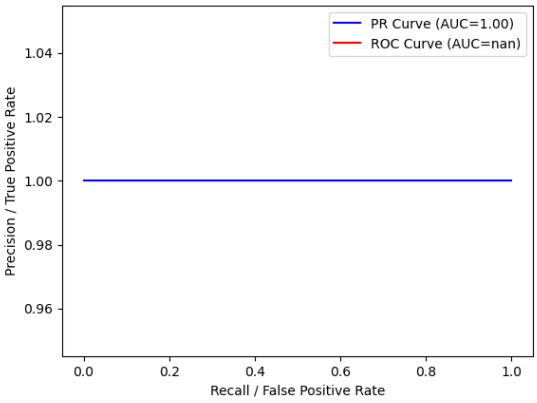
- **Accuracy**: It is a measure of how well the model predicts the correct class labels. It is calculated as the number of correct predictions divided by the total number of predictions.
- **Precision**: It is the ratio of true positive (TP) predictions to the total number of positive predictions. It measures the ability of the model to correctly identify positive instances.
- **Recall**: It is the ratio of true positive predictions to the total number of actual positive instances in the dataset. It measures the ability of the model to correctly identify all positive instances.

Performance Results



An area under the curve (AUC) of 1 for a precision-recall (PR) curve indicates that the classifier has achieved perfect precision and recall values on the given data, i.e., it has made no false positives and no false negatives. An AUC of 1 is the best possible score for a PR curve, and it indicates that the classifier is able to identify all positive samples correctly and has no false positives.
Thus, the ROC curve cannot be calculated, and the AUC is undefined or NaN.

```
/home/manisha/miniconda3/envs/myenv/lib/python3.8/site-packages/sklearn/metrics/
_ranking.py:990: UndefinedMetricWarning: No negative samples in y_true, false po
sitive value should be meaningless
  warnings.warn(
(myenv) manisha@manisha-HP-Laptop-15-bs1xx:~/Documents/Sem2/NS/TermProject/OurIm
pl/Approach_2/Scripts$ python3 tstMain.py
-----------First-----------
-----------fE1-----------
-----------fE2-----------
-----------Second-----------
-----------tM1-----------
Accuracy: 1.00
-----------tM2-----------
-----------Third-----------
Prediction for Brute-force attack starting
Prediction for Brute-force attack completed
Computing precision-recall curve and area under the curve
Computing ROC curve and area under the curve
/home/manisha/miniconda3/envs/myenv/lib/python3.8/site-packages/sklearn/metrics/
_ranking.py:990: UndefinedMetricWarning: No negative samples in y_true, false po
sitive value should be meaningless
  warnings.warn(
```



An accuracy score of 1 means that the model's predictions perfectly match the actual outcomes in the test data. However, a high accuracy score on the test set does not guarantee that the model will perform well on new, unseen data. So further rounds of testing will be required to improve performance of any model.

Conclusion

Thus from the two approaches we conclude that using the proposed method can help in detection of Brute-force attacks and thus generating alerts. This would help to reduce computational costs involved in decrypting the Brute-forced packets and hence reduces overhead involved.
Also on using our proposed ML-based classifier method can help to avoid setting any hardcoded thresholds for the detection and thus automating the process. Moreover, this also causes increase in precision and accuracy of the model.

Future Scope

- The current ML-based implementation can be extended to include other protocols like FTP, SSH, etc. as our current work is for HTTPS Traffic only.
- This implementation can also be extended to work on live Traffic as we work on stored Traffic Pcap only as a part of our implementation.
- This involves better integration between Python and Zeek such that Zeek based network monitoring & analysis capabilities can be used with Python scripts.
- Any other better performing ML Algorithm can be used in place of RandomForestClassifier algorithm which has been applied as part of our implementation. This can help to achieve a better performing model.
- This mechanism can be further implemented as a setup for real-world traffic for detction of Brute-force attacks in case of encrypted traffic.

References

- Base paper : Pascal Wichmann, Matthias Marx, Hannes Federrath, Mathias Fischer, “Detection of Brute-Force Attacks in End-to-End Encrypted Network Traffic”, ACM Digital Library, 2021.

- Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018.

- www.wikipedia.org
- https://cloud.smartdraw.com/editor.aspx?templateId=490dad73-de30-42bf-9a58-1789d56c1afd&flags=128#depoId=44970663&credID=-47961954
- https://tales-from-a-security-professional.com/intrusion-detection-system-have-they-become-useless-2ecf51488fed
- https://netacea.com/glossary/network-intrusion-detection-system-nids/#what-is-network-intrusion-detection-system-nids

Team Contribution

Manisha: Finalizing the base paper for project implementation; Collection for attack DataSet for the implementation; Implementation of the base paper method using zeek; Worked on integration of zeek with python in local system, local VM, virtual environment; Worked on implementation of python based modified approach, generated related logs and graphs; Involved in writing the report and the presentation making (Mid-Term & Final).

Madhuri: Went through available materials related to the project implementation; worked on Zeek implemenation and integration of zeek with python through pyzeek in local system, involved in part of presentation and report writing.

Anti-Plagiarism:

PLAGIARISM STATEMENT

I certify that this assignment/report is our own work, based on our personal study and/or research on our personal/lab equipment and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not
previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand our responsibility to report honor violations by other students if we become aware of it.

Names and Roll Nos: Manisha Mahapatra (CS22MTECH14009)
Julakuntla Madhuri (CS20BTECH11023)

Date: 27-04-2023

Signatures: Manisha Mahapatra, Julakuntla Madhuri