DATA SCIENCE-5163 Student Name: Amjad Alqahtani.
Date: 11/24/2024 ID: wkh221

Final Project:

Identifying Malicious Network Traffic Using Machine Learning

Introduction

Problem Statement

In the modern digital landscape, network traffic is constantly subjected to potential threats from malicious actors. Detecting such traffic is essential for safeguarding sensitive information and protecting the infrastructure. My project focuses on using machine learning techniques to detect anomalies in network traffic that may indicate malicious activity, such as cyberattacks like flood attacks.

Importance of Solving the Problem

Cyberattacks can cause significant damage, including financial losses and compromised infrastructure, affecting both organizations and individuals. Manually checking traffic is not practical given the vast amount of data on the web. Automated detection systems powered by machine learning are crucial for identifying and responding to cyber threats.

Benefits of Solving the Problem

This project will provide many benefits, particularly to organizations with high-security needs, such as financial institutions, hospitals, and government agencies. It will also support cybersecurity teams responsible for monitoring and defending against network intrusions, as well as individuals using personal devices who are vulnerable to network-based threats.

Literature Review

I reviewed three relevant studies to guide the design of this project, focusing on anomaly detection in network traffic.

Paper-1 **Elovici(2007)**: Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic

Summary of the paper

The paper proposes the eDare system, which operates on ISP infrastructure, such as switches and routers, to intercept and analyze network traffic for potential malicious content. Malicious code may either be known, based on previous detection history, or unknown. For known threats, eDare detects and blocks them immediately. For unknown threats, it uses machine learning algorithms to classify suspicious traffic, allowing the system to detect and prevent these threats in the future.

Architecture Design and Execution flow

First, if the threat is known, it will be directed to the known eThreat handling module. This module contains signatures of previously identified malicious code. When incoming traffic matches these signatures, the system blocks the malicious content without requiring analysis. Second, if the threat is unknown, it will be sent to the new eThreat detection module. This module uses machine learning algorithms to analyze suspicious files and detect potential malicious code. Various detection techniques are implemented as plug-ins (e.g., machine learning plug-ins). These plug-ins analyze the files and assign a threat score or rank. Then, the risk-weighting function aggregates the threat scores from the different plug-ins using a weighing system, which calculates an integrated rank to determine whether a file is malicious. Third, the collaborative module enables eDare to share threat detection results and warnings with other parts of the network. Fourth, the data stream manager collects clean network data after known threats have been filtered out by the known eThreat handling module. It extracts files from the continuous data stream and forwards them to the new eThreat detection module for further analysis. Finally, the entire system is managed and monitored through the eDare control center. The following figure illustrates the architecture of the eDare system

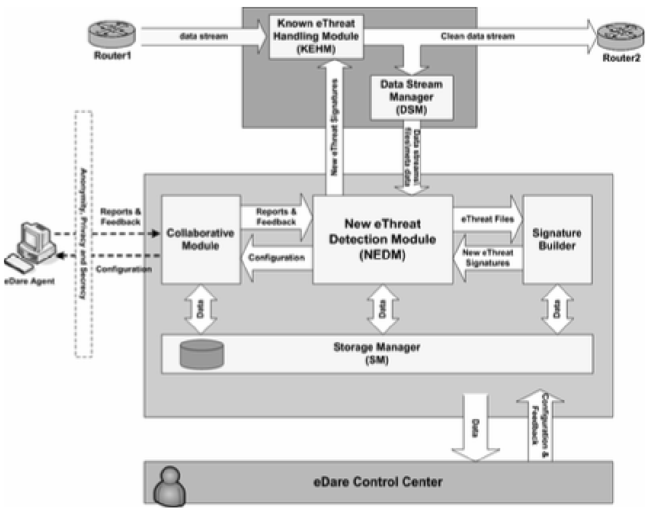


Fig. 1. Architecture of the eDare System

Methodology

In this section, I will delve into the new eThreat detection module, which is central to detecting unknown threats in the eDare system. This module is designed to use machine learning techniques as part of its detection process. Specifically, various machine learning algorithms are implemented as modular plug-ins within this module. These plug-ins analyze suspicious files to determine if they contain malicious code. The machine learning algorithms in this module include methods like Decision Trees, Bayesian Networks, and Artificial Neural Networks. Each of these techniques plays a specific role in classifying and identifying potential threats. The plug-ins generate threat scores based on their analysis, and these scores are then combined through a risk-weighting function, which produces a final threat rank. This approach allows the new eThreat detection module to adapt to evolving threats, making it highly effective in detecting previously unknown malicious code in network traffic.

Feature selection

1. N-grams: n-grams represent consecutive sequences of bytes from the binary representation of a file. The authors focused on 5-grams, which are sequences of five consecutive bytes. They selected the top 5,500 most frequent 5-grams. Then, they calculated a tf-idf score for each 5-gram. The tf-idf score: tf represents how often a particular 5-gram appears in a file, and idf indicates how rare that 5-gram is across the entire dataset of files. Finally, the Fisher Score was applied to rank the 5-grams by relevance, and the top 300 5-grams were selected for further analysis.

2. Portable Executable (PE) Format Features: is a standard binary format for windows executables (EXE or DLL files). The PE features include metadata such as file size, creation/modification time, machine type, linker version, section alignment, imported DLLs, and exported functions. These PE features are statically extracted using a specialized tool that parses the PE file structure to collect relevant information about the executable.
   By selecting only the top 300 5-grams and key PE features, the machine learning models were able to focus on the most informative aspects of the data, resulting in improved performance in detecting unknown malicious code.

Main experiment results and finding

Each plugin was trained using 30% of the dataset and tested on the remaining 70%, with the detection threshold set at 0.5. The experiment was deployed and tested in a network security lab under distinct clean and infected environments. A repository of 7,694 malicious files and 22,736 benign files was collected for evaluation. In this table, the summary of the experiments conducted on three machine learning algorithms (Decision Tree (DT), Artificial Neural Network (ANN), and Bayesian Network (BN)) is provided.

Classifier - Features Used	FPR	TPR	Accuracy
ANN (5 grams; top 300; Fisher)	0.038	0.893	0.945
BN (5 grams; top 300; Fisher)	0.206	0.885	0.816
BN (PE)	0.058	0.933	0.94
DT (5 grams; top 300; Fisher)	0.039	0.87	0.938
DT (PE)	0.035	0.925	0.955
Risk Weighting (Voting)	0.032	0.928	0.958

The False Positive Rate (FPR) is the proportion of non-malware files that are incorrectly classified as malware. The True Positive Rate (TPR) is the proportion of malware files that are correctly classified. Accuracy is the percentage of both malware and non-malware files that are correctly classified by the classifier.

Key Findings

Among individual plug-ins, DT (PE) (Decision Tree with Portable Executable features) had the highest accuracy at 95.5% and the lowest false positive rate at 3.5%. The overall system's performance was enhanced by using a Risk-Weighting (Voting) scheme, which combined the results of all plug-ins. This resulted in the highest accuracy of 95.8% and the lowest false positive rate of 3.2%.

Paper-2 **Jadav(2021)**: A Machine Learning Approach to Classify Network Traffic

Summary of the paper

The growing complexity of network environments, including encrypted networks like Tor and VPN, has made it challenging to monitor and classify traffic, especially for detecting malicious activity. Traditional methods, such as protocol analyzers and deep packet inspection, have proven insufficient. This paper leverages machine learning to address these challenges, utilizing the CIC-Darknet 2020 dataset to classify benign and darknet traffic. Key techniques like SMOTE for balancing and PCA for dimensionality reduction are applied, followed by a comparison of ensemble methods, logistic regression, tree-based classifiers, and Naive Bayes. The models are evaluated using metrics such as Accuracy, Precision, Recall, F1-Score, and MCC.

Methodology

Data Preprocessing: The raw CIC-Darknet 2020 dataset is preprocessed, addressing class imbalance using SMOTE (Synthetic Minority Over-sampling Technique) to balance the benign and darknet traffic. PCA (Principal Component Analysis) is applied to reduce the dimensionality of the dataset. The dataset contains 141,530 rows and 85 features. PCA reduces the number of features from 85 to approximately 25 features, which explain 90% of the variance in the dataset. After applying SMOTE (Synthetic Minority Over-sampling Technique), the class imbalance is addressed, and the dataset is balanced with both the benign and darknet traffic classes containing 117,219 samples each.

Feature Extraction: Important network features such as IP addresses, port numbers, protocol types, and packet lengths are identified and extracted. These features are standardized to normalize the data, allowing the models to perform better on a balanced and simplified feature set.

Modeling and Classification: several machine learning models were trained and evaluated, including ensemble methods (e.g., AdaBoost, Random Forest, Bagging, Extra Trees, and Gradient Boosting), along with logistic regression, tree-based classifiers, and Naive Bayes. The ensemble methods, particularly Extra Trees and Decision Trees, delivered the best results, with both train and test accuracies reaching 99% to 100%.

These top-performing models also achieved an F1-Score and Matthews Correlation Coefficient (MCC) close to 0.99, demonstrating excellent classification performance. In contrast, models like Logistic Regression and Naive Bayes performed significantly worse, with accuracy and MCC values ranging from 0.46 to 0.54 for Logistic Regression and Gaussian Naive Bayes.

Evaluation: The models are assessed using key metrics such as Accuracy, Precision, Recall, F1-Score, and Matthews Correlation Coefficient (MCC) to ensure a thorough performance comparison.

Results

Accuracy:

Extra Trees and Decision Trees are both achieved 99% to 100% accuracy.
Logistic Regression: around 71%
Naive Bayes (Gaussian): 54%.

Precision:

Extra Trees and Decision Trees: 99% precision.
Logistic Regression: 67%.
Naive Bayes (Gaussian): 52%.

Recall:

Extra Trees and Decision Trees: 99% recall.
Logistic Regression: 84%.
Naive Bayes (Gaussian): 89%.

F1-Score:

Extra Trees and Decision Trees: 99% F1-Score.
Logistic Regression: 75%.
Naive Bayes (Gaussian): 66%.

Matthews Correlation Coefficient (MCC):

Extra Trees and Decision Trees: 0.99 MCC.
Logistic Regression: 0.46.
Naive Bayes (Gaussian): 0.13.

Paper-3 **de Lucia(2019)**: Detection of Encrypted Malicious Network Traffic using Machine Learning

Summary of the paper

This paper addresses the challenge of detecting malicious activity within encrypted network traffic, particularly traffic encrypted using Transport Layer Security (TLS). Traditional detection methods such as pattern matching and deep packet inspection are becoming ineffective. Malware authors have also adopted encryption, making it harder to detect malicious communications. The paper classifies malicious and benign traffic without needing to decrypt it using machine learning models called Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel and a Convolutional Neural Network (CNN). Instead of relying on content, these models use TLS record sizes and direction as key features.

Methodology

Data Preprocessing:

The dataset used in the study consists of TLS-encrypted network traffic, including both normal traffic and malware traffic. The TLS features (record size, type, and direction) are extracted from packet capture files. Each traffic flow is split into bidirectional flows, and the TLS record size and direction are key features used for classification. The direction is encoded as a positive value for server and negative for client, and only application and handshake record types are used.

Architecture

Two models are proposed: SVM (Support Vector Machine): A non-linear SVM using a Radial Basis Function (RBF) kernel. CNN (Convolutional Neural Network): A one-dimensional CNN that treats the sequence of TLS record sizes and directions as a time series. The model uses several convolutional and pooling layers followed by dense layers, aiming to detect patterns in the traffic data that indicate malicious behavior.

SVM Model:

The SVM model uses the frequency of unique TLS record sizes as input features. These features are stored in a hash vector. The SVM is trained using a sparse hash vector, with only non-zero values representing the frequency of unique TLS record sizes in a session. This method effectively detects malicious encrypted traffic.

CNN Model:

The CNN model processes the sequence of TLS record sizes as a time series. It uses a one-dimensional vector with record sizes and directions (positive for server, negative for client) as input. The CNN architecture consists of several layers: Two convolutional layers with filters and activation functions (ReLU). Pooling layers to down-sample the data. Dense layers for final classification.

Results

The dataset was split into 70% for training and 30% for testing, with a stratified 10-fold cross-validation to maintain class balance. The classifiers were evaluated using several metrics, including Accuracy, Precision, Recall, F1-score, and False Positive Rate (FPR).

Classifier	Accuracy	Precision	Recall	F1-score	False Positive Rate
Support Vector Machine (SVM)	99.97%	99.98%	99.94%	99.97%	0.02%
Convolutional Neural Network	99.91%	99.93%	99.87%	99.91%	0.07%

Both models performed exceptionally well, but the SVM outperformed the CNN in all metrics, particularly achieving a lower FPR. The SVM model's strong performance indicates it is more reliable in detecting malicious encrypted traffic with fewer false alarms compared to the CNN.

Dataset

Dataset Overview

Name: Network Traffic Data: Malicious Activity Detection
Source: Kaggle Dataset
https://www.kaggle.com/datasets/advaitnmenon/network-traffic-data-malicious-activity-detection
Size: 8 columns with a still-to-be-verified number of rows.
Input Features:

- Timestamp: Time when the packet was captured.
- Source IP Address: Sender's IP address.
- Destination IP Address: Receiver's IP address.
- Source Port: Communication port used by the sender.
- Destination Port: Communication port used by the receiver.
- Protocol: Network protocol (e.g., TCP, UDP).
- Length: Size of the packet.

Target Variable:

- bad_packet: A binary label (1 = malicious packet, 0 = normal packet).

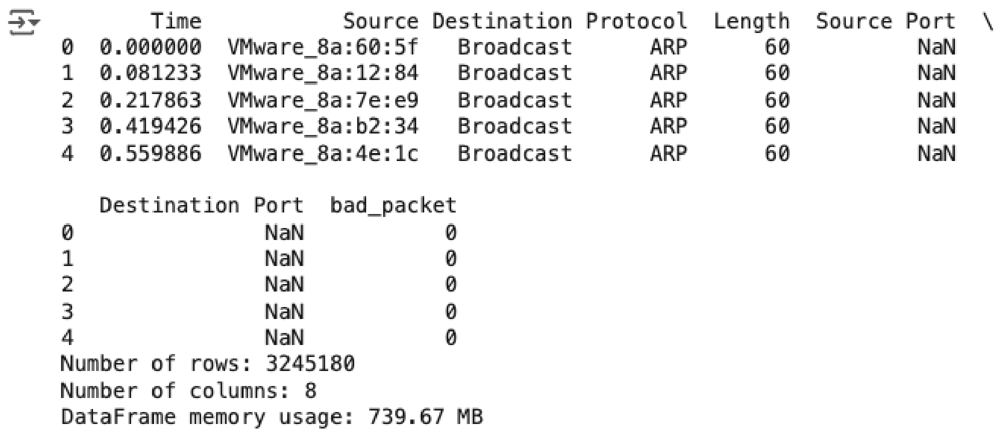
I explored and preprocessed this dataset by visualizing distributions of numerical features (e.g., packet length) using histograms and analyzing categorical features (e.g., protocols) with bar charts.

Data Description

This dataset has 8 columns, with missing values in Source Port and Destination Port. The bad_packet column is a target variable, indicating whether a packet is malicious (1) or normal (0). Features like Source, Destination, and Protocol are categorical, while Time, Length, and port columns are numerical. Preprocessing will be required to handle missing values and encode categorical features for modeling. Categorical features require encoding for machine learning models, while numerical features are typically scaled or normalized for better model performance. Categorical features often need to be transformed into numerical form using techniques like one-hot encoding, label encoding, or hash encoding.

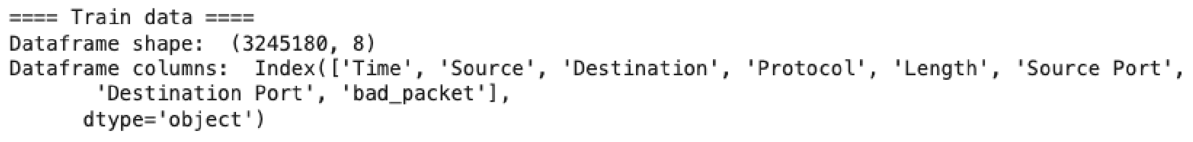
Data Preprocessing

Data Loading and Display



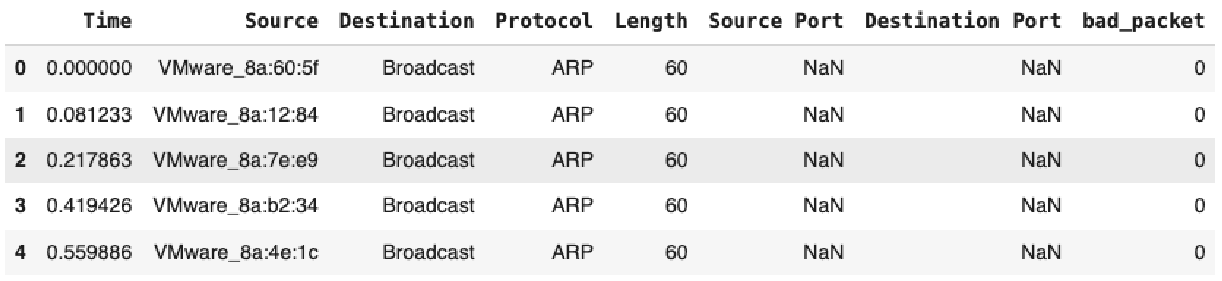
```
         Time           Source Destination Protocol  Length  Source Port  \
0  0.000000  VMware_8a:60:5f   Broadcast      ARP      60          NaN   
1  0.081233  VMware_8a:12:84   Broadcast      ARP      60          NaN   
2  0.217863  VMware_8a:7e:e9   Broadcast      ARP      60          NaN   
3  0.419426  VMware_8a:b2:34   Broadcast      ARP      60          NaN   
4  0.559886  VMware_8a:4e:1c   Broadcast      ARP      60          NaN   

   Destination Port  bad_packet  
0               NaN           0  
1               NaN           0  
2               NaN           0  
3               NaN           0  
4               NaN           0  
Number of rows: 3245180
Number of columns: 8
DataFrame memory usage: 739.67 MB
```

Data shape and dataset instruction



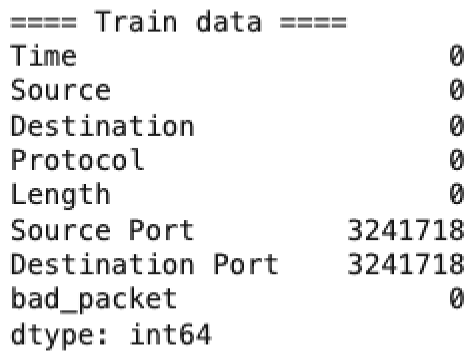
```
==== Train data ====
Dataframe shape:  (3245180, 8)
Dataframe columns:  Index(['Time', 'Source', 'Destination', 'Protocol', 'Length', 'Source Port',
       'Destination Port', 'bad_packet'],
      dtype='object')
```

Overview of the dataset structure is shown in the figure below:



	Time	Source	Destination	Protocol	Length	Source Port	Destination Port	bad_packet
0	0.000000	VMware_8a:60:5f	Broadcast	ARP	60	NaN	NaN	0
1	0.081233	VMware_8a:12:84	Broadcast	ARP	60	NaN	NaN	0
2	0.217863	VMware_8a:7e:e9	Broadcast	ARP	60	NaN	NaN	0
3	0.419426	VMware_8a:b2:34	Broadcast	ARP	60	NaN	NaN	0
4	0.559886	VMware_8a:4e:1c	Broadcast	ARP	60	NaN	NaN	0

Missing Values

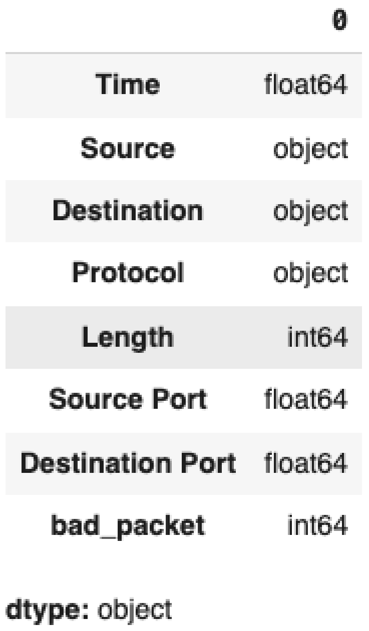


```
==== Train data ====
Time                    0
Source                  0
Destination             0
Protocol                0
Length                  0
Source Port       3241718
Destination Port  3241718
bad_packet              0
dtype: int64
```

From the above figure, we can understand that almost all source and destination ports are empty and have no value of being features of our ML models.

Data Analysis

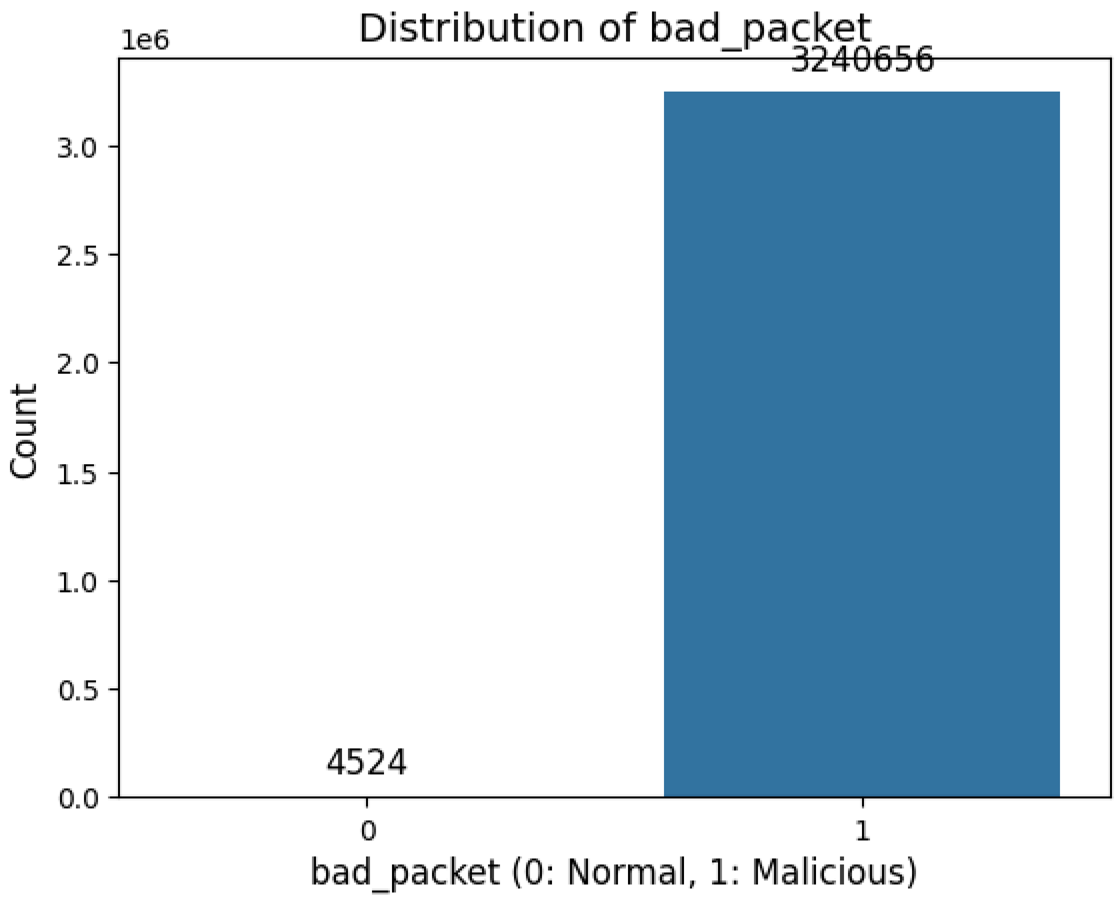
Data types



	0
Time	float64
Source	object
Destination	object
Protocol	object
Length	int64
Source Port	float64
Destination Port	float64
bad_packet	int64

dtype: object

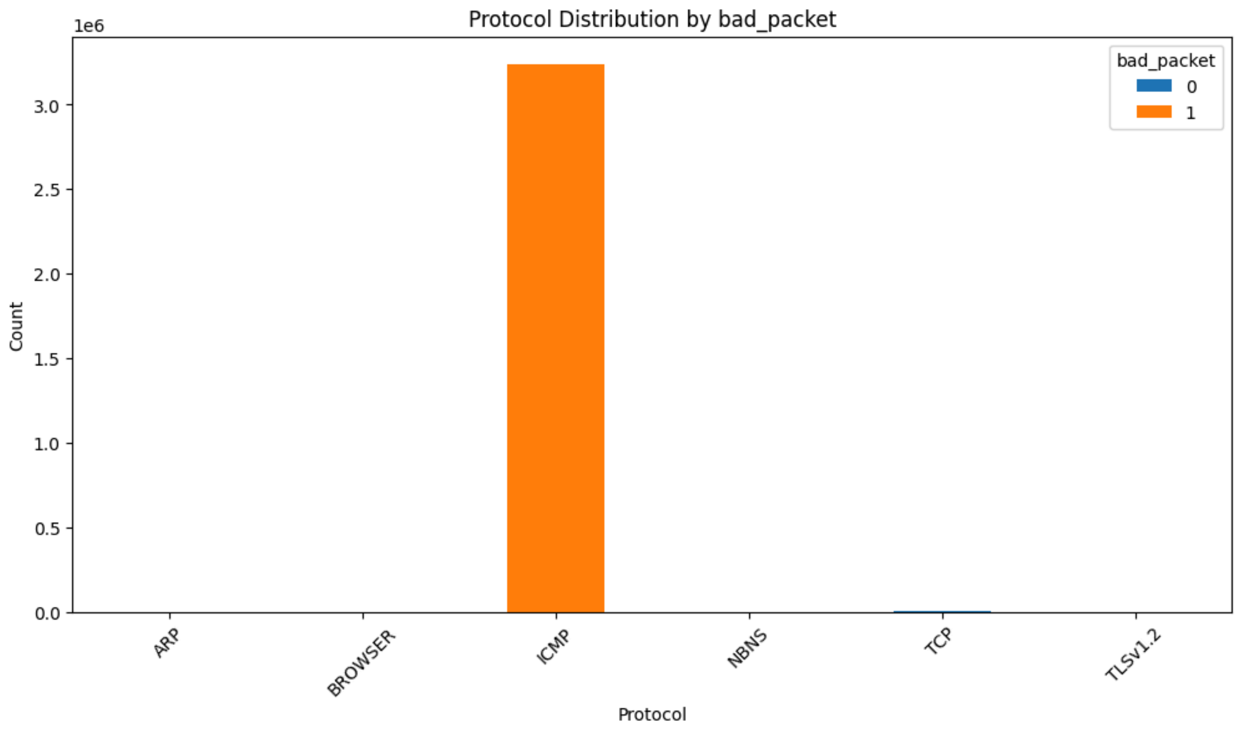
<u>Distribution of target value</u>



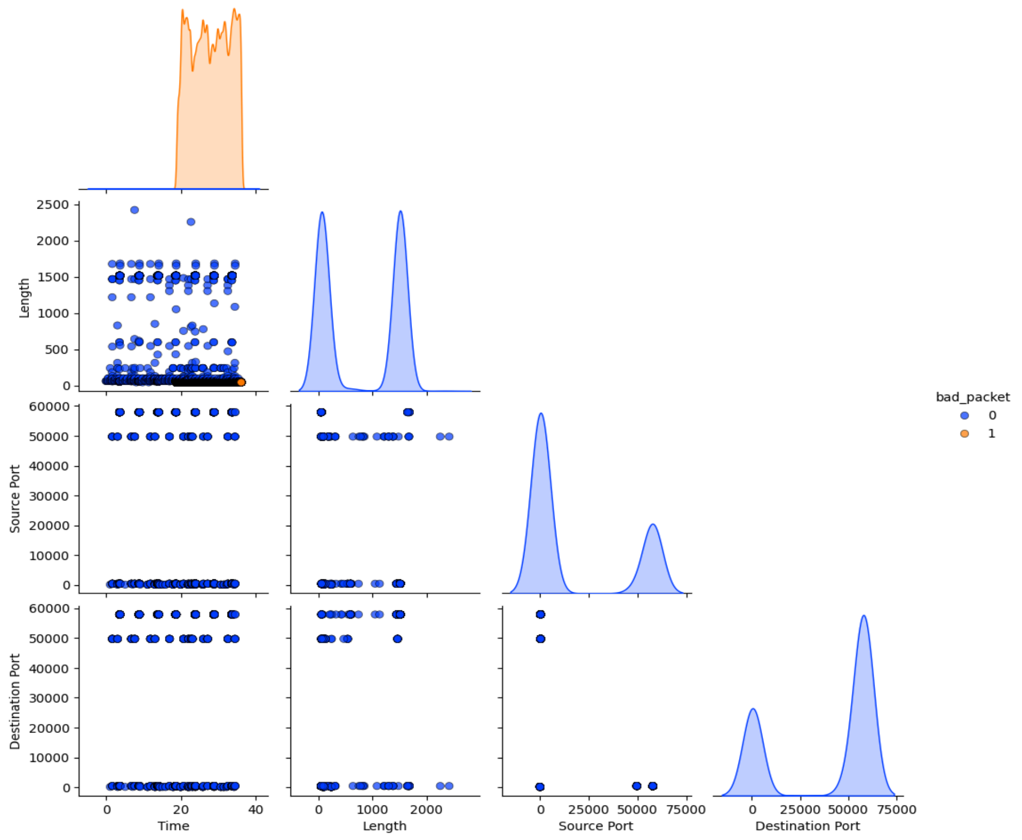
<u>Distribution of protocol feature and its unique values:</u>

Number of unique protocols: 6
Unique protocols: ['ARP' 'ICMP' 'BROWSER' 'TLSv1.2' 'TCP' 'NBNS']
Index(['Time', 'Source', 'Destination', 'Protocol', 'Length', 'Source Port',
    'Destination Port', 'bad_packet'],
   dtype='object')

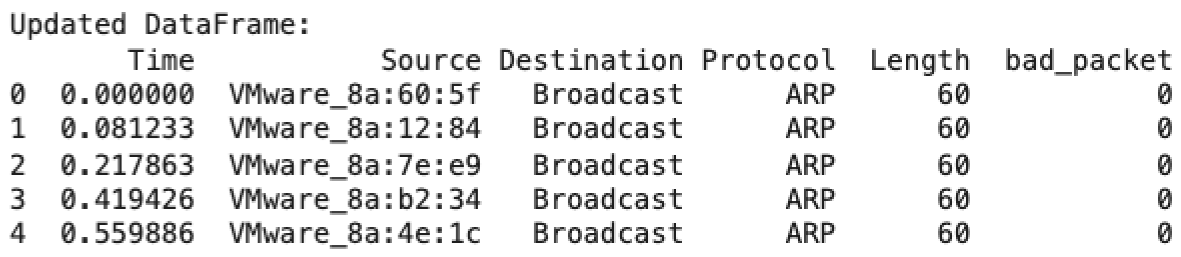
```
bad_packet     0        1
Protocol
ARP          241        0
BROWSER       47        0
ICMP         821  3240656
NBNS          53        0
TCP         2974        0
TLSv1.2      388        0
```



Distribution analysis using pairplot



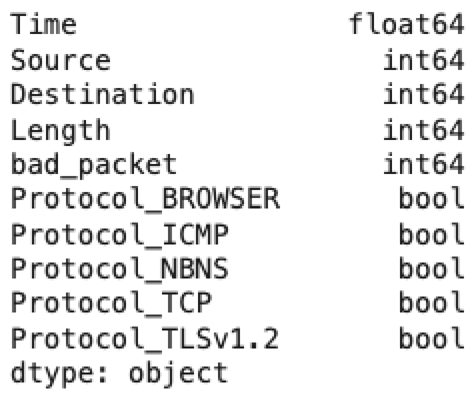
Dataset after remove useless features:



```
Updated DataFrame:
       Time          Source Destination Protocol  Length  bad_packet
0  0.000000  VMware_8a:60:5f   Broadcast      ARP      60           0
1  0.081233  VMware_8a:12:84   Broadcast      ARP      60           0
2  0.217863  VMware_8a:7e:e9   Broadcast      ARP      60           0
3  0.419426  VMware_8a:b2:34   Broadcast      ARP      60           0
4  0.559886  VMware_8a:4e:1c   Broadcast      ARP      60           0
```

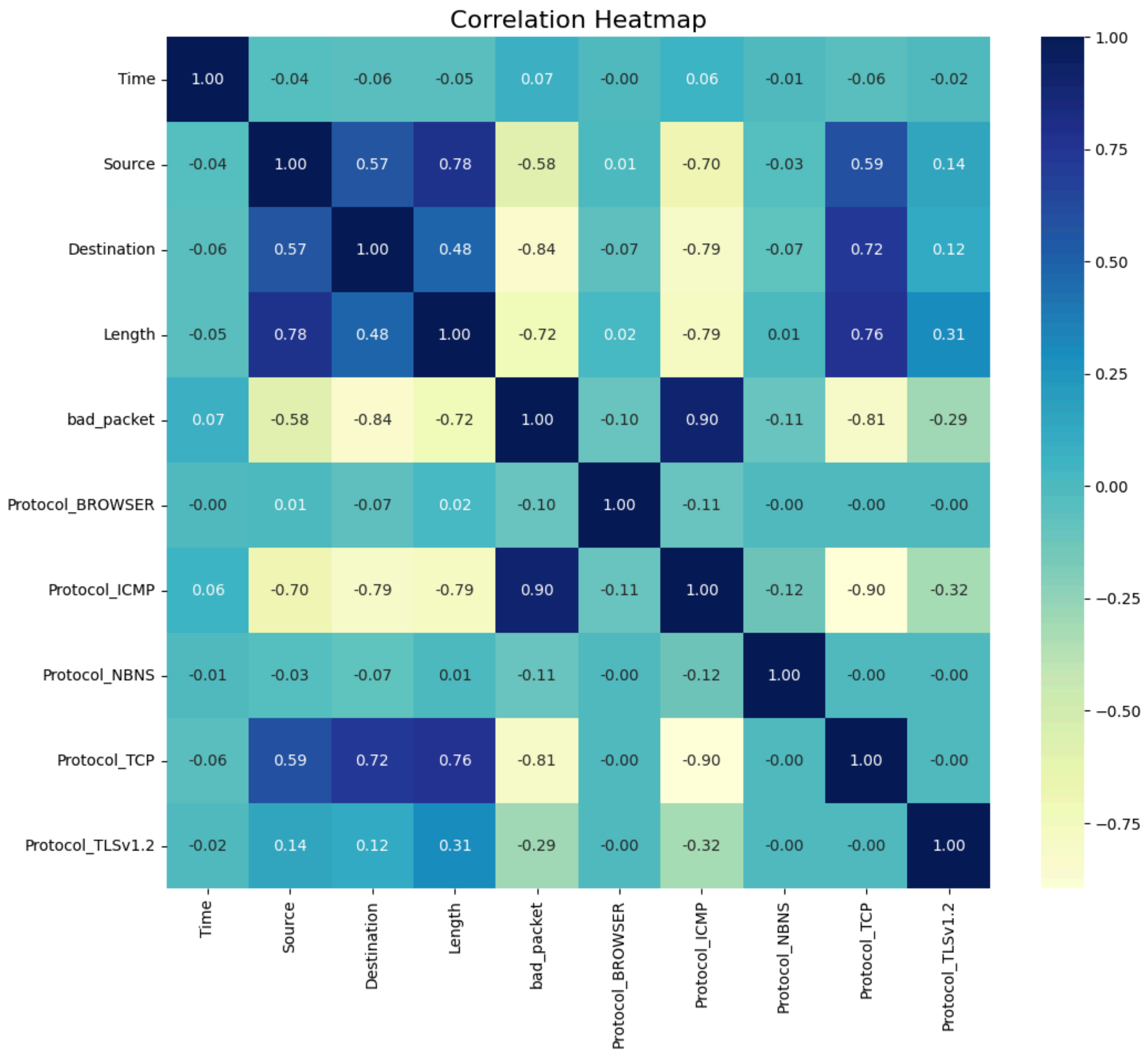
Convert categorical columns into numerical

This will be for (source and destination address):



```
Time                 float64
Source                 int64
Destination            int64
Length                 int64
bad_packet             int64
Protocol_BROWSER        bool
Protocol_ICMP           bool
Protocol_NBNS           bool
Protocol_TCP            bool
Protocol_TLSv1.2        bool
dtype: object
```

Correlation heatmap



1. Time feature has (3,245,180 unique values): Every entry has a unique timestamp. I derive time-based patterns like cyclic features (sin/cos transformations) from it.
2. Source (29 unique values): I encode this feature using label encoding.
3. Destination (9 unique values): categorical feature
4. Protocol (6 unique values): A small set of unique values indicating a strongly categorical feature. One-hot encoding works well.
5. Length (67 unique values): A numerical feature with moderate variability. It can be scaled using standardization to improve model performance.
6. Source Port and Destination Port (6 unique values each): almost missing values.

Data Preparation

Splitting dataset

I split the dataset to be 70% as training data and 30% for testing data.

```
Training set size: 2271626 samples
Test set size: 973554 samples
```

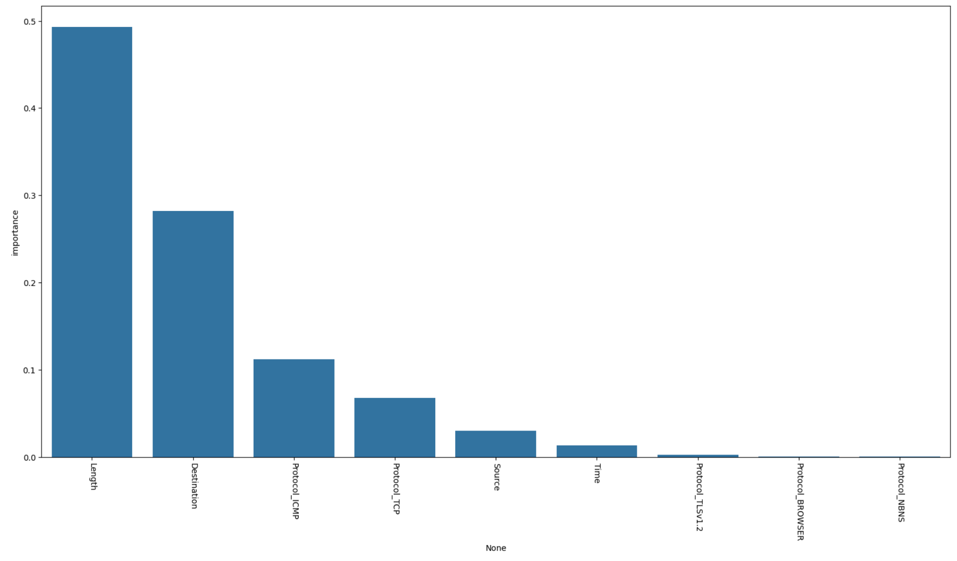
Then, I checked the balance:

```
Train set distribution:
bad_packet
1    0.998601
0    0.001399
Name: proportion, dtype: float64

Validation set distribution:
bad_packet
1    0.998617
0    0.001383
Name: proportion, dtype: float64
```

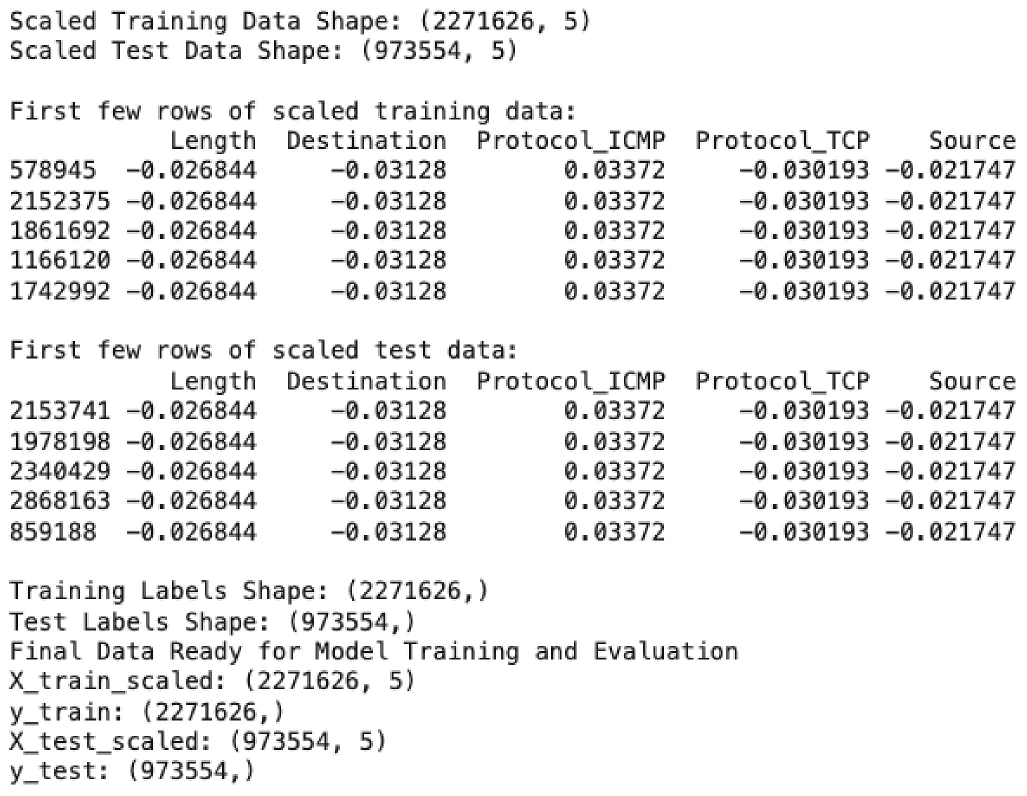
Data Scaling

Scaling the data is important to avoid that some features will have a bigger impact on the model training than others. This is especially important when we are dealing with features that have different units of measure.

Feature importance using Random Forest Classifier:



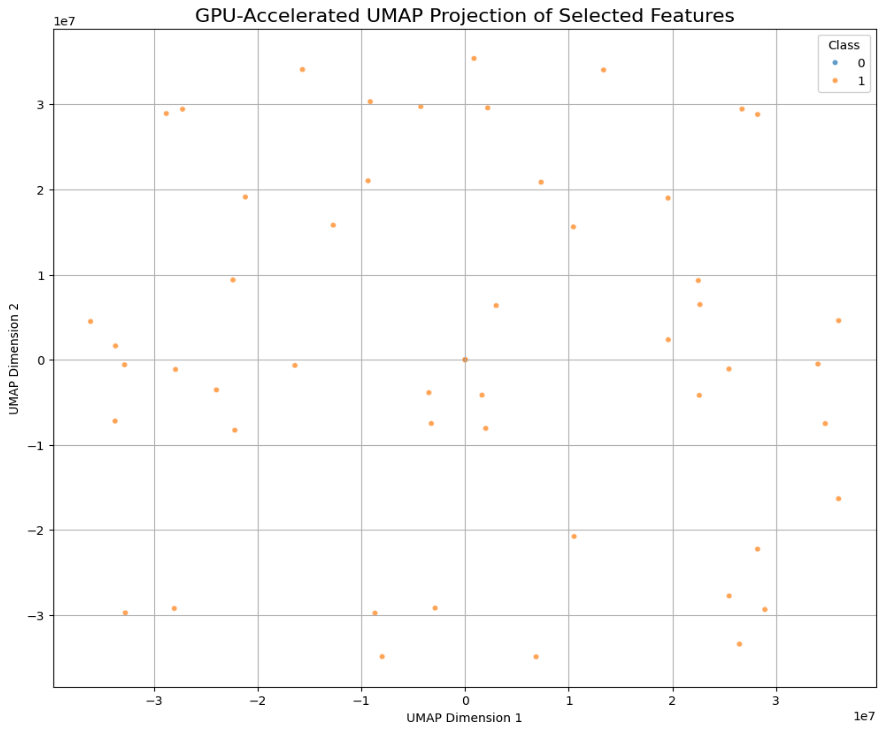
I have selected the most important features:
Index(['Length', 'Destination', 'Protocol_ICMP', 'Protocol_TCP', 'Source'], dtype='object')
Final shape and structure of the dataset after doing scaling, normalization, and eliminating useless features:



```
Scaled Training Data Shape: (2271626, 5)
Scaled Test Data Shape: (973554, 5)

First few rows of scaled training data:
           Length  Destination  Protocol_ICMP  Protocol_TCP    Source
578945  -0.026844     -0.03128        0.03372     -0.030193 -0.021747
2152375 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1861692 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1166120 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1742992 -0.026844     -0.03128        0.03372     -0.030193 -0.021747

First few rows of scaled test data:
           Length  Destination  Protocol_ICMP  Protocol_TCP    Source
2153741 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1978198 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
2340429 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
2868163 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
859188  -0.026844     -0.03128        0.03372     -0.030193 -0.021747

Training Labels Shape: (2271626,)
Test Labels Shape: (973554,)
Final Data Ready for Model Training and Evaluation
X_train_scaled: (2271626, 5)
y_train: (2271626,)
X_test_scaled: (973554, 5)
y_test: (973554,)
```

UMAP visualization

Reduce data dimensionality to 2 dimensions and plot the data using matplotlib.



Verify Feature Selections:

```
Scaled Training Data Shape: (2271626, 5)
Scaled Test Data Shape: (973554, 5)

First few rows of scaled training data:
           Length  Destination  Protocol_ICMP  Protocol_TCP    Source
578945  -0.026844     -0.03128        0.03372     -0.030193 -0.021747
2152375 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1861692 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1166120 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1742992 -0.026844     -0.03128        0.03372     -0.030193 -0.021747

First few rows of scaled test data:
           Length  Destination  Protocol_ICMP  Protocol_TCP    Source
2153741 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
1978198 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
2340429 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
2868163 -0.026844     -0.03128        0.03372     -0.030193 -0.021747
859188  -0.026844     -0.03128        0.03372     -0.030193 -0.021747

Training Labels Shape: (2271626,)
Test Labels Shape: (973554,)
Final Data Ready for Model Training and Evaluation
X_train_scaled: (2271626, 5)
y_train: (2271626,)
X_test_scaled: (973554, 5)
y_test: (973554,)
```

Data Training

Model Training In this section we will train different models and compare their results. We will use the following models: KNN, SVC, and Random Forest Classifier.

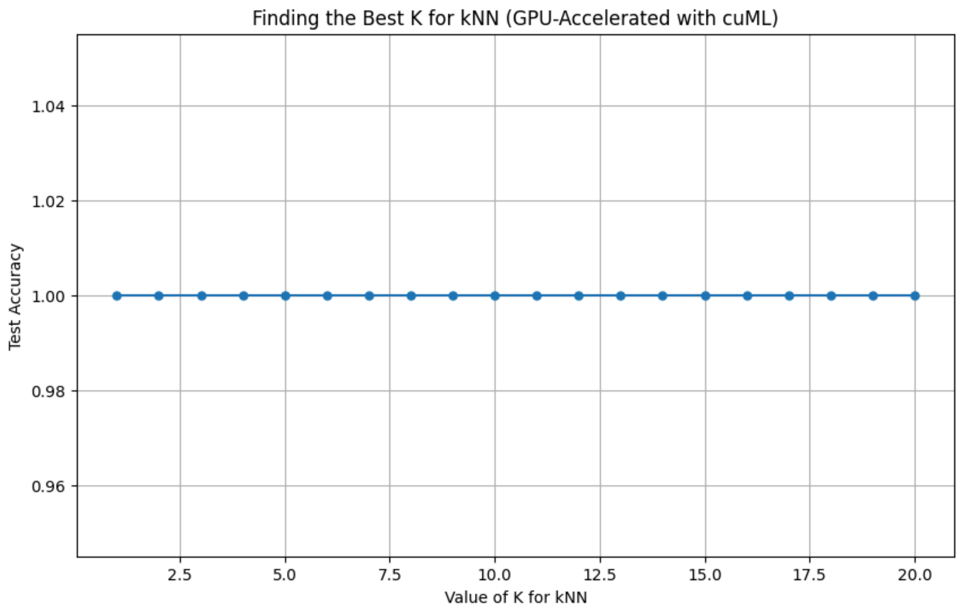
KNN is a simple, non-parametric algorithm that performs well when there are clear patterns in the dataset, such as clusters of similar behavior (e.g., normal vs. malicious traffic).

SVC handles complex decision boundaries, which are often present in cybersecurity problems where malicious traffic might subtly overlap with normal traffic.

RFC excels with datasets containing a mix of numerical and categorical features, such as your protocols and packet lengths. It can automatically handle feature selection by identifying the most important predictors.

K-Nearest Neighbors (KNN)

Finding the best K hyperparameter, I will use the validation set to find K value. I will then use this K value to train the model on the training set and evaluate it on the test set.



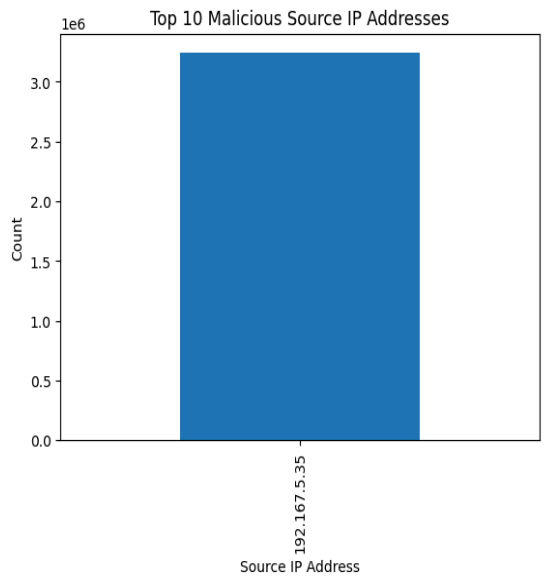
Success Metrics

- Accuracy: The ratio of correctly classified packets to the total number of packets. Accuracy is important for giving an overall measure of model performance.
- Precision: The ratio of correctly identified malicious packets to the total identified as malicious. This is critical given the imbalanced nature of the dataset (i.e., fewer malicious packets).
- Recall: The ability of the model to correctly identify all malicious packets.
- F1 Score: The harmonic mean of precision and recall, offering a balanced measure when class distribution is uneven.

```
Best k: 1
Best accuracy: 1.0000

Classification Report for Best K:
             precision    recall  f1-score   support

           0       1.00      1.00      1.00      1346
           1       1.00      1.00      1.00    972208

    accuracy                           1.00    973554
   macro avg       1.00      1.00      1.00    973554
weighted avg       1.00      1.00      1.00    973554
```

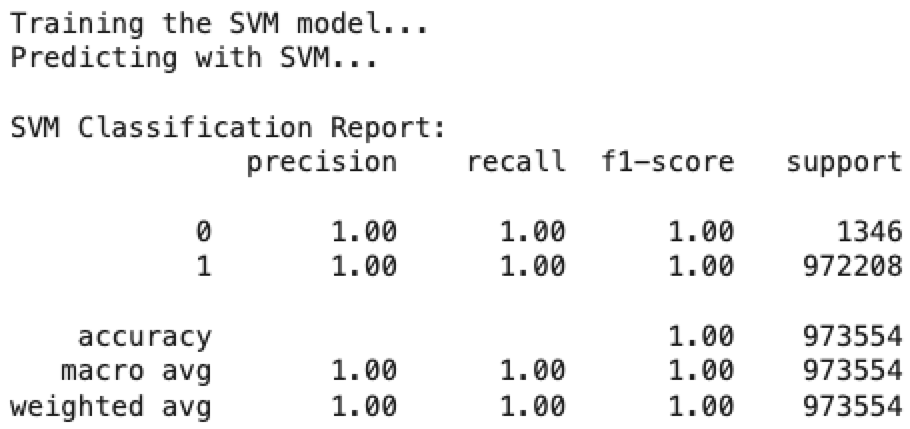
The model has achieved the best result in all matrics.
Why? This is because there is only one dominant source address that generates all malicious traffic, and the rest of sources are all normal. So, the ML was able to understand this result and ended up with perfect scores in all matrics.

```
Source
192.167.5.35    3240656
Name: count, dtype: int64
Source
192.232.16.204     2235
192.167.5.35       1034
192.167.5.22        786
142.251.32.14       128
VMware_8a:7e:e9      36
VMware_8a:12:84      35
VMware_8a:60:5f      35
VMware_8a:31:5e      35
VMware_8a:4e:1c      35
VMware_8a:b2:34      35
192.167.1.235        12
VMware_8a:25:c0      10
192.167.4.164         9
192.167.6.52          9
192.167.9.41          9
192.167.9.105         9
192.167.7.108         9
192.167.5.247         9
192.167.9.39          9
VMware_8a:9d:21       7
192.167.2.126         6
192.167.3.73          6
192.167.4.188         6
192.167.9.21          5
VMware_8a:0a:66       4
VMware_8a:7f:3f       4
VMware_8a:aa:27       3
VMware_8a:5c:e6       2
```



Support Vector Classifier(SVC)

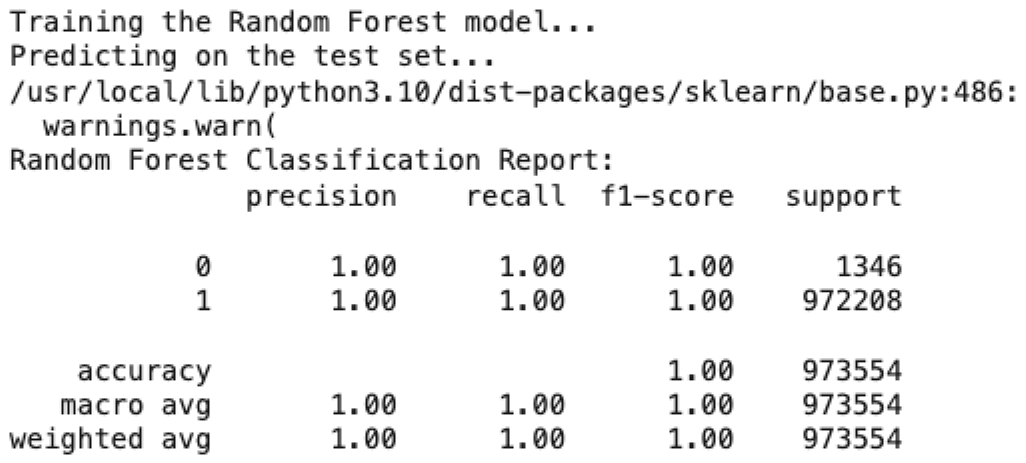
I apply the SVM on my data and the result is perfect given the same reason in KNN.



```
Training the SVM model...
Predicting with SVM...

SVM Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1346
           1       1.00      1.00      1.00    972208

    accuracy                           1.00    973554
   macro avg       1.00      1.00      1.00    973554
weighted avg       1.00      1.00      1.00    973554
```

Random Forest Classifier

I apply the RFC on my data and the result is perfect given the same reason in KNN.



```
Training the Random Forest model...
Predicting on the test set...
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486:
  warnings.warn(
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1346
           1       1.00      1.00      1.00    972208

    accuracy                           1.00    973554
   macro avg       1.00      1.00      1.00    973554
weighted avg       1.00      1.00      1.00    973554
```

Conclusion

In this project, I used machine learning techniques to address the critical challenge of detecting malicious network traffic. Through comprehensive data preprocessing and feature selection, I prepared a dataset that accurately reflects the complexities of network traffic.

Various machine learning models, including K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Random Forest Classifier (RFC), were trained and evaluated on the dataset. All models achieved exceptional performance metrics, such as accuracy, precision, recall, and F1 score, primarily due to the dataset's identifiable patterns. The findings highlight the importance of feature engineering and appropriate model selection in achieving robust results.

By automating the detection of malicious traffic, this project demonstrates the practical benefits of machine learning in enhancing network security, offering organizations an efficient and scalable solution to mitigate cyber threats.

Future work could focus on incorporating more complex and diverse datasets, evaluating real-time detection capabilities, and applying advanced models such as deep learning to improve adaptability to new and evolving threats.

Overall, this project underscores the potential of machine learning to transform cybersecurity practices and protect critical infrastructure from malicious activities.

Reference

Elovici, Y., Shabtai, A., Moskovitch, R., Tahan, G., & Glezer, C. (2007). Applying machine learning techniques for detection of malicious code in network traffic. In J. Hertzberg, M. Beetz, & R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence (Vol. 4667, pp. 44-50). Springer. https://doi.org/10.1007/978-3-540-74565-5_5

Jadav, N., Dutta, N., Deva Sarma, H. K., Pricop, E., & Tanwar, S. (2021). A machine learning approach to classify network traffic. 2021 International Conference on Electronics, Communications and Information Technology (ECAI), 1-6. https://doi.org/10.1109/ECAI52376.2021.9515039

de Lucia, M. J., & Cotton, C. (2019). Detection of encrypted malicious network traffic using machine learning. 2019 IEEE Military Communications Conference (MILCOM). IEEE. https://doi.org/10.1109/MILCOM47813.2019.9020856